TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

BING LIU

FACE DETECTION IN IMAGES BY TRANSFORM CLASSIFIERS

Master of Science thesis

Examiner: prof. Irek Defée
Examiner and topic approved by the
Faculty Council of the Faculty of
Electric and Computer Engineering
on 4th March 2015

ABSTRACT

BING LIU: Face detection in images by transform classifiers
Tampere University of Technology
Master of Science Thesis, 48 pages
June 2015
Master's Degree Programme in Information Technology
Major: Multimedia
Examiner: Professor Irek Defée
Keywords: face detect, LBP, Haar fetures, integral image, AdaBoost, classifier cascade

The thesis describes operation of classifiers based on Haar and LBP features and evaluates their performance on selected data sets. Haar and LBP based classifiers are popular in machine learning due to their good performance and fast operation after training. These classifiers were developed over the years and their structure is now quite complicated due to the Adaboost cascade which is used to adaptive optimize the classification. The classifier problem is studied in the thesis for the face detection in images. Classifiers are trained on image sets with faces in different positions and image sets with no faces. Afterward performance in detecting faces is evaluated with test data sets. Two type of image sets are utilized during detection. First set is made by real faces from available database. Second set is made by faces which were generated by 3D graphics software called Facegen. Both data type were used in experiments separately and in mixture. We found that the Haar classifier performs better than LBP but this is at the cost of increased computation time. For real faces the performance is very good, over 90% correct face detection. For 3D generated faces the performance is much weaker, 65% correct face detection. When both data sets are mixed the performance drops according to the proportion of both types of faces. This indicates that 3D faces are much more difficult to detect by the classifiers. Exact reason for this could not be established it could be due to the lack of small features in 3D faces, greater similarity of images or some other issue which could be the topic of further research.

PREFACE

This thesis is about face detection by classifiers based on the Adaboost method with Haar and LBP features. The Adaboost classifier is employed in face detection and recognition area frequently. Typically real human face images are used to train the classifier, moreover both 3D faces which are generated by 3D graphics software and the real faces were utilized in our experiments. The results of experiments proved that the classifier trained by real face provided better detection results than 3D graphics faces. I appreciate much the guidance of my supervisor professor Irek Defée who gave me many ideas how to improve the results for 3D face training.

Tampere, 15.5.2015

Bing Liu

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

LBP	Local Binary Pattern
Adaboost	Adaptive Boost
API	Application Programming Interface
ROR	Rotation of Texture
PEC	Personal Event Collection Data Set
FDDB	Face Detection Data Set and Benchmark Database

1. INTRODUCTION

Face detection is the problem of finding if there are faces in images and evaluation of their parameters. Face detection is a basis of face identification or recognition which used in large number of applications, such as digital cameras, security and human-machine interaction. All kind of face information extraction is routinely performed by people and appears quite simple but it is hard to understand how exactly human brain does it. Research on face detection acquired many distinct algorithms. Image face detection is challenging since algorithms must be efficient enough [2]. Generally there exist four different methods to implement the face detection : knowledge based method, invariant feature based method, template matching based method and machine learning based method [1].

This thesis focuses on one type of machine learning methods for face detection based on transform classifiers. Machine learning is a popular topic at present. Distinguished from other detection methods machine learning requires a large amount of training samples to acquire sufficient number of features. The process of training is also called learning phase. The sample collection is of considerable importance in face detection. It is essential to collect wide diversity of samples due to the complexity of face poses in real life. Besides the rapid response of detection acts a critical role in the real time detection The thesis is concentrated on the investigation of face detection using real face training data and compare the efficiency of detection using faces generated with a 3D graphics software.

In the thesis we used two different methods based on machine learning concepts. These methods employ special transforms based on Haar features and Local Binary Patterns (LBP). The approach based on Haar features is robust and quick method for performing face detection. but requires huge computation processing which is challenging. Special method called integral image has been develop to reduce the computational burden make this method practical.

Second method is based on the LBP transform features. The LBP feature is calculated by comparing pixel value with the values of its surrounding pixels. Calculation of LBP is much simpler than Haar and it does not require special methods. In consequence the LBP is much faster than Haar but its performance in face detection may not be as good.

The learning phase in face detection consists of providing to the system a large number positive images with face and negative images with no faces. This requires vary large number images, at least several thousands since to achieve the proper detection, the

learning processing should catch enough features to represent the face. Typically more than ten thousands features may be needed to create machine learning system with good performance. The training used for processing features is the procedure called adaptive boosting. The training system tries to optimize the classification result by selecting the set of features. This is done by using weak and strong classifiers and selecting best of them. In a weak classifier human face can be represented by only one feature. However the classifier with only one feature will produce many error results. To improve the performance of detection multiple weak classifiers can be combined. The thesis will focus on the problem of optimizing the classification by adaptive boosting method.

The thesis is organized as follows. In Chapter 2 the Haar and LBP features are described with efficient methods for their calculation. Chapter 3 has detailed description of classifiers and training structures including weak and strong classifiers, the Adaboost method and cascading. In Chapter 4 the experimental environment is described including the real face data sets and method of collecting face samples using 3D graphics software. Results of training and performance evaluation are provided. The thesis ends with Conclusions in which the results and future directions are summarized.

2. FEATURES

This chapter describes two types of transforms which generate features for machine learning based face detection system used e.g. in the OpenCV software. The first are Haar-like features named so since they are similar to Haar transforms. The second are Local Binary Patterns (LBP) features .

2.1 Haar-like Features

Haar feature is a rectangular block of certain size which is divided into two types of sub-blocks. Pixels within one type sub-block are summed up and then differences between sums for the sub-blocks of different types are taken. Example sub-blocks are shown in *Figure 2.1,* with the meaning that black sub-block(s) are subtracted from the white sub-block(s). There can be very many types of blocks and sub-blocks. Shape of blocks and sub-blocks will decide about the ability to describe certain features within it.

The earliest Haar-like feature was defined by Papageorgious et al. [4]. The features Viola and Jones used in their fast face recognition are of three different types with five different arrangements [3]: features with two rectangular sub-blocks, features with three rectangular sub-blocks and feature with four rectangular sub-blocks as shown in *Figure 2.1*.

Figure 2.1 *The basis Haar-like features*

Lienhart and Maydt defined an extended set of features in which the rectangular features rotated by 45 degrees were proposed [5]. This extended set of features primarily includes four categories: edge features, line features, center-surround features and diagonal features as shown in *Figure 2.2*. This set of features includes all features in *Figure 2.1* except the last one .

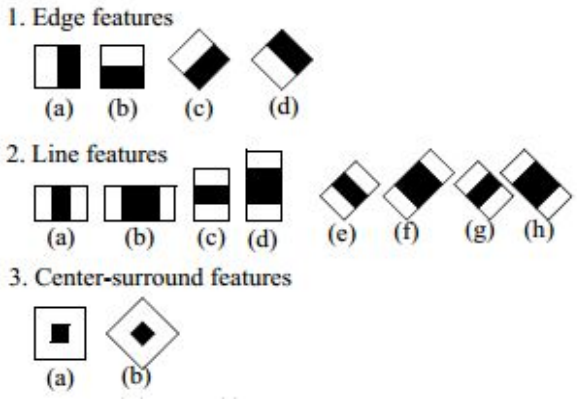

Figure 2.2 *Different types of Haar like features. Image source from [5]*

For the method of Viola and Jones, it was proved that face detection result could be [3] with less than 1% false positive rate although only two pairs of Haar features were adapted.

With the low false rate it demonstrates that Haar feature is robust and effective for face detection. Generally it is enough for face detection to use upright Haar rectangles. With the cost of more explicit classification result the face detection would have considerable benefit on time consumed if the processing is without rotation features. In our experiments, the basis features were primarily implemented.

Haar features can be used to represent parts of the face. For example eyebrows are darker than surrounding area and the center feature shown in *Figure 2.3* will be sensitive to it. Similarly the feature on the right in *Figure 2.3* will be sensitive to the nose area.

Figure 2.3 *Different feature can represent distinct part of face*

2.1.1 Representation of Haar-like Features

Assume a sub window covers $W \times H$ pixels in the image as in *Figure 2.4*. The sub window or rectangle can be represented [5] by 5 elements:

$$r = (x,y,w,h,\alpha) \text{ with } x,y \geq 0, x+w \leq W, y+h \leq H, w,h \geq 0 \text{ and } \alpha \in \{0^{\circ}, 45^{\circ}\} \quad (2.1)$$

In formula 2.1 *(x,y)* is the first left corner coordinate of the rectangle, *w* and *h* is the respectively width and height of rectangle, α is the rotation phase of rectangle which contains two values 0 and 45 due to the construction of Haar rectangles.

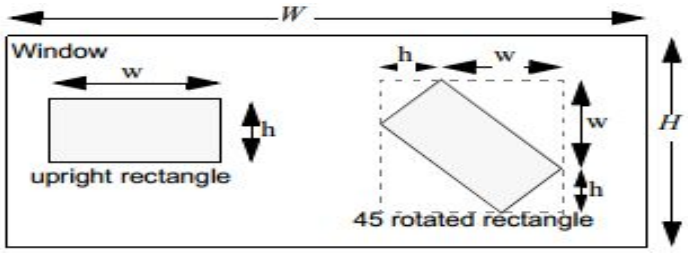

Figure 2.4 *Example of an upright rectangle and 45°rotated rectangle*

Theoretically a Haar feature includes less than four rectangles but in special cases it can be bigger. The value of feature with *k* rectangles can be expressed [2,5] by formula 2.1:

$$feature_I = \sum_{i=1}^{k} w^{(i)}.u^{(i)} \text{ or } feature_I = \sum_{i \in I=\{1,\ldots,N\}} \omega_i \cdot \mathrm{Re}cSum(r_i) \qquad (2.2)$$

where $w^{(i)}$ or ω_i is the weight and $u^{(i)}$ is the intensity of the covered area for *i-th* rectangle. *N* is the number of rectangle. Formula 2.3 defines the relationship of the weight.

$$\sum_{i=1}^{k} w^{(i)} = 0 \qquad (2.3)$$

The detail interaction of weights in Haar features is displayed in *Figure 2.5*. A Haar feature contains two rectangles with one weight of which is -1.

Figure 2.5 *The rectangle weights*

Generally the weight of rectangle does not need to be a fixed number of 1 or -1. Under the condition of formula 2.3 the weight value could be random responding to the area of rectangles as formula 2.4.

$$\frac{s_1}{s_2} = -\frac{\omega_2}{\omega_1} \qquad (2.4)$$

where s_1 and s_2 is the area of a given rectangles and ω_1, ω_2 are its relevant weights. But in order to provide the simplicity of calculations the value of weights in the transform classifiers is defined to be +1 or -1.

Calculation of Haar features is thus done with very simple operations of additions and subtractions. But there is a large diversity of Haar features as size increases the amount of features in certain sub-window. An increasing amount of features could supply a better detection result but also requires more complicated computing. The amount of calculations required is shown next.

2.1.2 The Number of Haar-like Features in Sub-window

Previously two types of features were illustrated, upright and rotated. The upright ones are mainly applied in feature calculation, there are more than 160,000 such features in sub-window with the size of 24×24. If a classifier includes both upright and rotated features their number will be quite large. Here we show how to calculate the number of features in both cases. The method of calculation for both types of features is slightly different and will be introduced separately.

Assume there is sub window with width W and height H, the number of upright feature in the sub window is then [5] equivalent to

$$N_0 = xy \cdot (W+1-w\frac{x+1}{2})(H+1-h\frac{y+1}{2}) \tag{2.5}$$

where w and h is the minimum value for width and height of Haar like features which theoretically can be 1 x 1. The scaling factor $x= [W/w]$, $y = [H/h]$ is the maximum value along x and y directions. From the formula 2.5 the maximum of this number strongly depends on the size of minimum Haar feature and the size of sub-window. Typically in the detection tasks the size of the searching window which represents the face is not very small while in practice the size of minimum Haar feature is slightly greater than one.

The computing of number of rotated features must consider the rotation angle. For rotated features the w and h in scale x and y are not the real width and height of rectangle but the horizontal and vertical distance along x and y direction. Supposed we have the same sub-window, the number of rotated Haar features is calculated as:

$$N_{45} = xy \cdot (W+1-z\frac{x+1}{2})(H+1-z\frac{y+1}{2}) \text{ with } z = w+h \tag{2.6}$$

For example if the minimum feature size is $w=1, h=2$ and window size $W = 24, H = 24,$ the number of features for upright and rotated cases are respectively 21,600 and 4232. As can be seen the number of feature is by far bigger than the number of pixels.

The sub-window size plays important role in training a classifier that if the size is relatively large the workload for computing the feature will be heavy. In turn if the size of sub-window is quite small then the number of features is smaller but the detection result would be inferior. So a appropriate sub-window size should be selected for the best compromise between the detection performance and computing speed.

2.2 Integral Image

As shown above even for a small sub-window a large amount of features is produced. If calculated directly for each sub-window this would considerably slow down the pace of calculation. For dealing with this problem an integral image method was developed. This method achieves big reduction in the number of calculations for the successive rectangles in image.

Instead of calculating the sum for each rectangle area for original image separately, the integral image method substitutes the pixel value in the original image by the sum of pixel values in rectangle and forms a representation of original image called integral image. The *Figure 2.6 (a)* shows the original image with its pixel values and *(b)* illustrates the integral image with integral values. The integral transformation is defined as in the formula 2.7 and 2.8

$$s(x\text{-}1,y) = i(x\text{-}1,y\text{-}1)+i(x\text{-}1,y) \text{ and } s(x,y\text{-}1) = i(x\text{-}1,y\text{-}1)+i(x,y\text{-}1) \quad (2.7)$$

$$s(x,y)=s(x\text{-}1,y)+s(x,y\text{-}1)-s(x\text{-}1,y\text{-}1)+i(x,y) \quad (2.8)$$

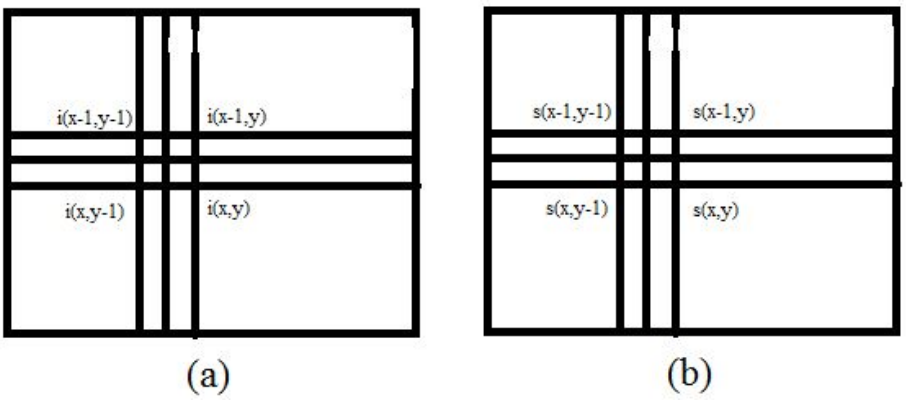

Figure 2.6 *(a) original image (b) integral image*

i(x,y) and *s(x,y)* in formula respectively represent pixel information in the original image and summed value in integral image. The formula 2.7 represents the summed value along the horizontal and vertical directions. The integral transformation executes in the block with size of 2×2 so *s(x,y)* is the last pixel in such block. In the continuous calculation the previous *s(x,y)* will remain unchanged. For the integral formula *s(x-1,y-1)*

has the same calculation principle as *s(x,y)*. In addition integral value with minus index will be set to value of zero. *Figure 2.7* depicts the calculation of integral image.

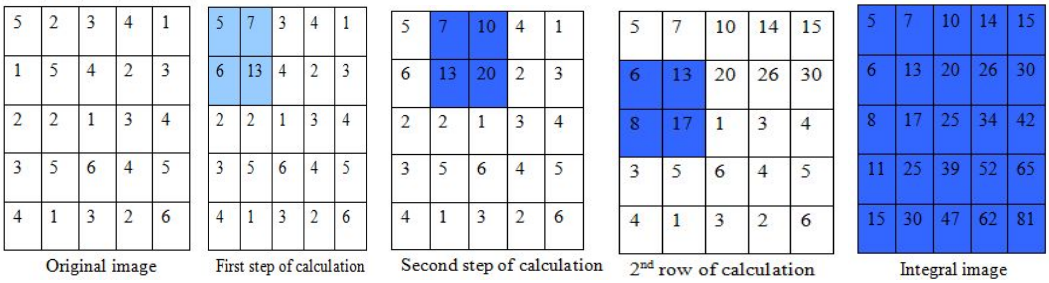

Figure 2.7 *The details of integral image calculation*

With the integral image the processing of summing up would be effective and rapid. In *Figure 2.8* the yellow area in *(a)* represents the summed area in original image and (*b)* indicate the location of pixel associated with summed area in integral map. It can be thus seen that the representation of the summed area table (SAT) in the original image is converted into the calculations based on four elements representing rectangles.

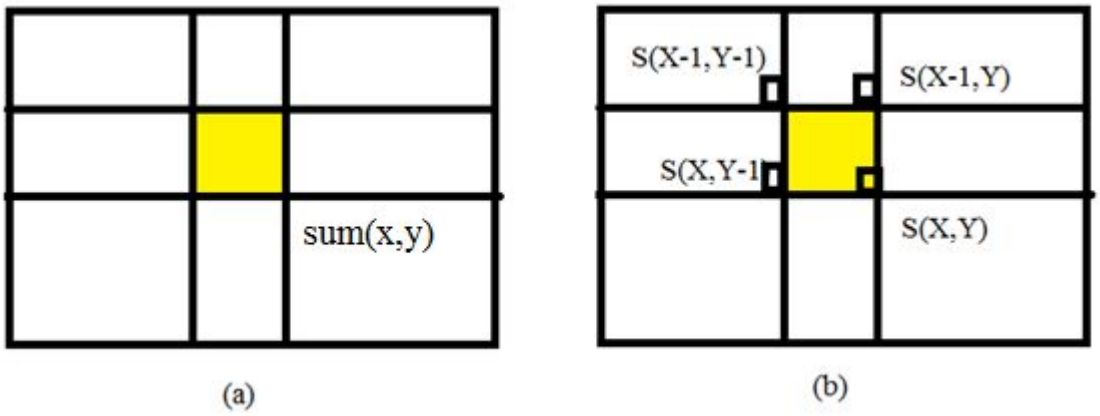

Figure 2.8 *(a) the summed area in original image (b) the summed area in integral image*

Directly from the definition of summed area the sum in original image is as:

$$Sum(x,y) = \sum_{x \le X, y \le Y} i(x,y) \tag{2.9}$$

where *i(x,y)* is the pixel value *and X* and *Y* are the last coordinates for the summed area. Sum in the original image is actually traditional addition which sums up all pixel values in the region of summed area. The same summed area in the integral map is calculated as:

$$SumInter(x,y) = S(x,y) + S(x-1,y-1) - S(x-1,y) - S(x,y-1) \tag{2.10}$$

S(x,y) is the integral value in the original image. *Figure 2.8* describes the typical arrangement of the summed area. There are also special conditions for summed area such as the corner location and border location as in *Figure 2.9.*

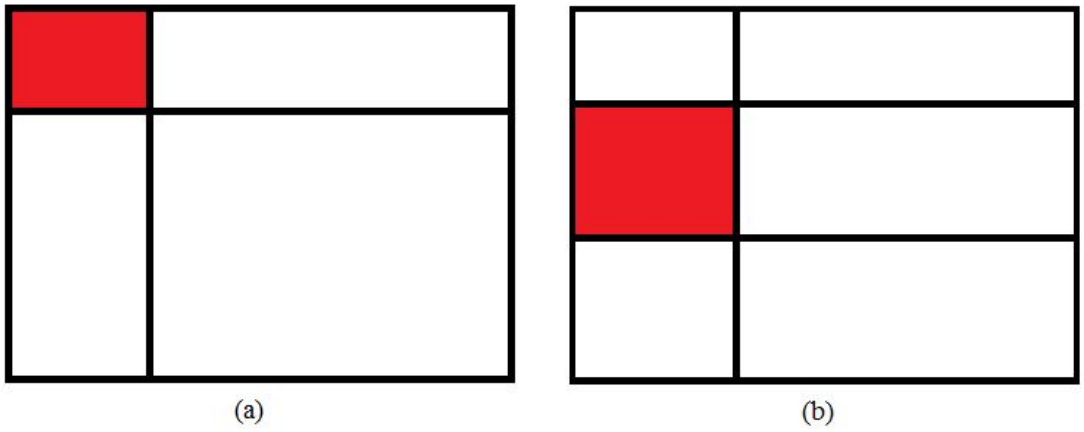

Figure 2.9 *(a)corner condition of summed area (b) border condition of summed area*

Form *Figure 2.9* in the corner condition except *s(x,y)* the other values of (2.10) are missing. So the missing value can be substituted by the value of zero, in the other words in this situation single value of *s(x,y)* is able to represent the summed value. Similarly the summed value of border condition can be represented by the remaining two values. *Figure 2.10* illustrates the details of comparison for summed values in the corner and normal condition.

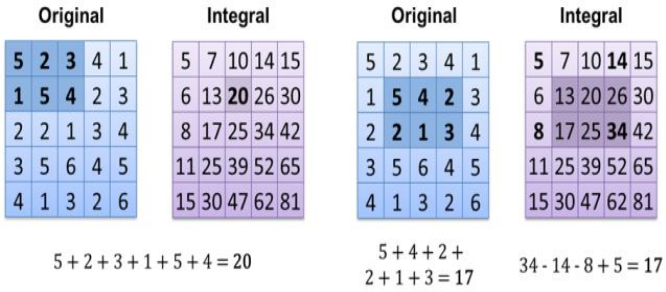

Figure 2.10 *left 2 shows the corner condition with 3×3 table and right 2 draws the situation in general*

Similarly as in the upright integral image, the rotated feature with 45° integral image is defined as:

$$RSUM(x,y) = \sum_{y' \le y, y \le y - |x - x'|} i(x', y') \tag{2.11}$$

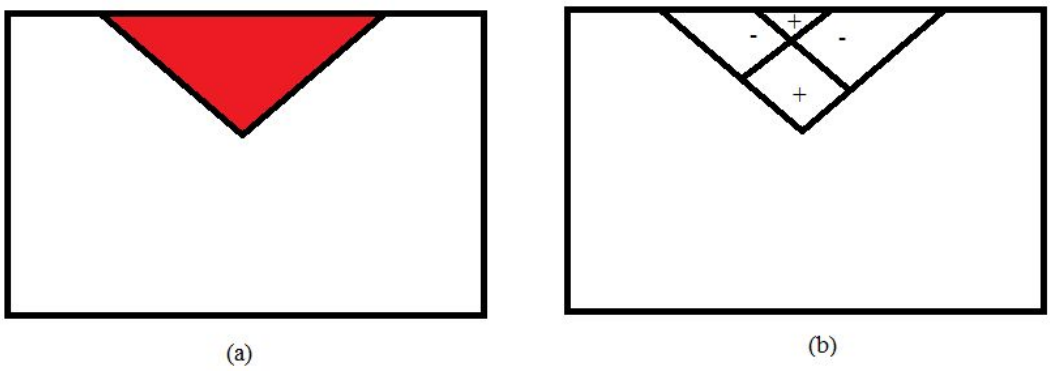

Figure 2.11 *(a) 45 degree feature in original image (b)sign relationship in integral image*

In formula 2.11 *RSUM(x,y)* is the summed area in the red area of *Figure 2.11(a)*. *x'* and *y'* is the coordinate in the inverted triangle's vertex. The weight of rotated Haar like feature in integral image is indicated as in *Figure 2.11(b)*. The integral image for rotated feature is different from the one for upright features but the method of integral image has similarities for both cases.

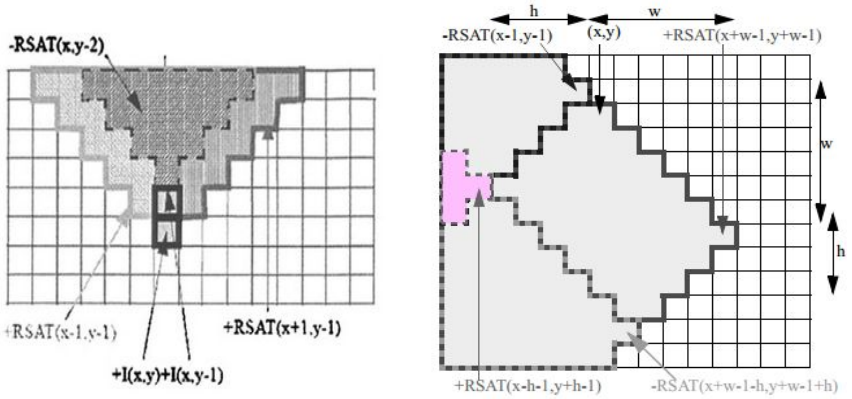

Figure 2.12 *(left)The integral image RS(x,y) for rotated feature (right)the summed value for (x,y) rectangle by rotated integral image[5]*

By the position relationship in *Figure 2.12*, the integral image *RSUM(x,y)* for the area starting from *(x,y)* can be defined [5] as:

$$RSUM(x,y) = RSUM(x-1,y-1) + RSUM(x+1,y-1) + I(x,y) + I(x,y-1) - RSUM(x,y-2) \quad (2.12)$$

With

$$RSUM(-1,y) = RSUM(x,-1) = RSUM(x,-2) = 0 \text{ and } RSUM(-1,-1) = RSUM(-1,-2) = 0$$

By going through the complete original image along row or column direction can get the rotated integral image for the whole image. Supposed there is rotated rectangle feature $r = (x,y,w,h,45)$ as shown in *Figure 2.12(right)*. The summed value for such

rectangle is starting from *(x,y)* with 45 degree rotation. Similar as the upright rectangle summed value, the rotated rectangle summed value can be determined as :

$$RecSum(r_i) = RSUM(x-h-1,y+h-1)+RSUM(x+w-1,y+w-1)-$$
$$RSUM(x+w-1-h,y+w-1+h)-RSUM(x-1,y-1) \quad (2.13)$$

thus no matter if the feature is upright or rotated the integral image has capacity to convert the calculation of all pixel values into the addition of four elements. With the increase of feature size the efficiency of integral image would be even more prominent.

2.3 Local Binary Pattern

Local Binary Pattern (LBP) is developed by Ojala et al. [9] to represent the texture of objects. In image domain texture is the basic element and texture primarily includes base texture and the sequence of base textures. Base texture describes the primitive of objects' information. In the other words the element in an image is comprised of a sequence of base textures. The name local binary pattern means representation of comparison result between the pixel and its local surrounding pixels. The center of pixel is regarded as threshold to compare with neighboring pixels. In the illustration in *Figure 2.13* if the current pixel is bigger than the center one then assign 1 to the binary sequence, else set it to 0. The transformation of LBP is defined in formula 2.14

$$LBP(x_c, y_c) = \sum_{p=0}^{p-1} 2^p s(i_p - i_c) \quad (2.14)$$

where (x_c, y_c) is the coordinate of center value in the LBP map and p is the number of pixels encompassing the center pixel. $s(x)$ represents the sign function and i_c is the center value of the pattern. *Figure 2.13* depicts the common LBP searching window with the size of 3×3. The calculation orientation should be clockwise and the LBP map can be constructed with scanning the entire image pixel by pixel.

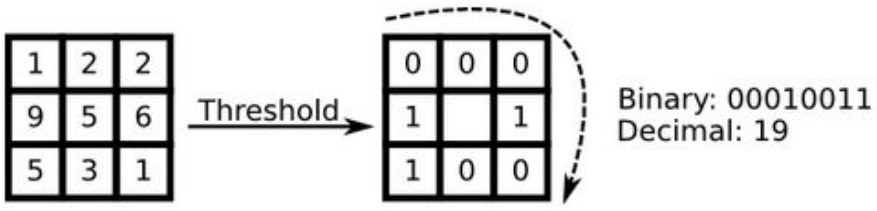

Figure 2.13 *The processing of LBP*

It is also possible to increase the size of 3×3 of the searching window of LBP. In the general the LBP window has radius R and amount of surrounding pixels P. Briefly

radius of LBP window denotes the distance of center pixel to others. Now the formula 2.14 can be evolved into formula 2.15

$$LBP_{R,P}(x_c, y_c) = \sum_{p=0}^{p-1} 2^p s(i_p - i_c) \tag{2.15}$$

In the window, P and R affects significantly the texture detail of LBP map. With the increasing of P the particulars of image will be more apparent, in other words the more pixel information the more details will be retained. Similarly the texture details are reduced by enlarging the radius R since the further there are two pixels the more the distinction between the pixels value will be. Three different situations for P and R are displayed in *Figure 2.14*.

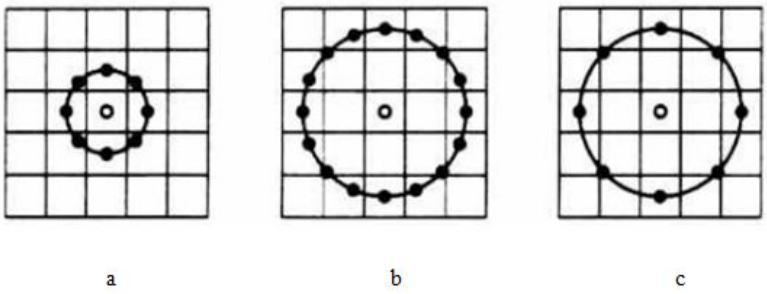

Figure 2.14 *(a) shows the simplest model of LBP with window size of 3×3 and 8 neighborhoods with radius of 1.(b) and (c) are the extension of basic LBP respectively with P=16 , R= 3 and P=8, R=2*

Figure 2.14(a) shows the LBP window with P=8 and R = 1 and the pixels drop straight onto the coordinates of original image. In the case of straight projection, the evolution of LBP transformation according to formula 2.15 is demonstrated in *Figure 2.15.*

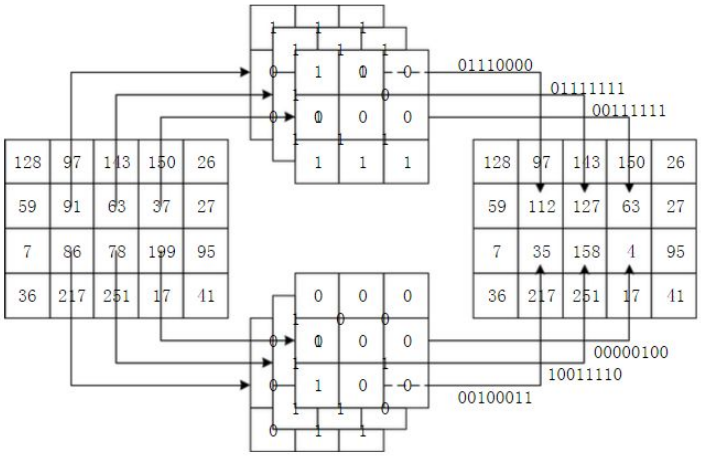

Figure 2.15 *LBP feature mapping with straight projection*

In conditions (b) and (c) in *Figure 2.14* some pixels are in the intersection of coordinate system of original image.The pixel value which is not in the position of pixel such as the cross point in the figure can be evaluated by formula 2.16:

$$x_{p'} = x_c + R\cos(\frac{2\pi p'}{p}) \text{ and } y_{p'} = y_c - R\cos(\frac{2\pi p'}{p}) \qquad p' \in [1,\ldots,p] \qquad (2.16)$$

p is the total number of surrounding pixels in the window and p' is the index of pixel. $x_{p'}$ and $y_{p'}$ is the iteration at p'. *Figure 2.16* shows the construction of crossing pixel *f(x,y)* and the pixel value in the intersection of coordinate approximates the linear combination of four independent partials. The pixel value is visible at coordinate (0,0),(0,1),(1,0) and (1,1).

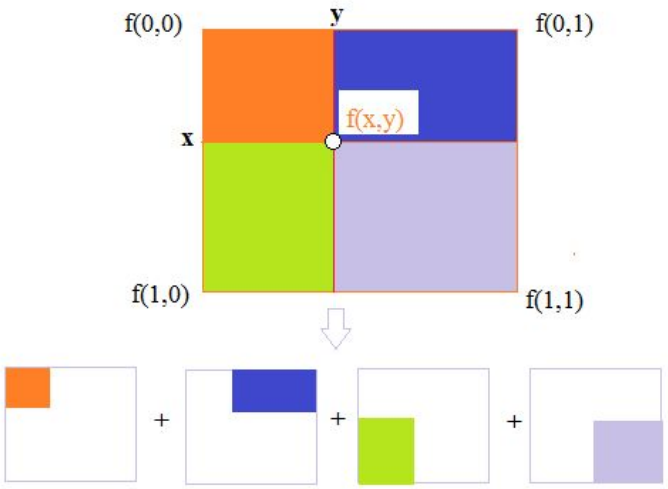

Figure 2.16 *The construction of cross pixel*

Representation of *f(x,y)* is described by formula 2.17

$$f(x,y) \approx f(0,0)(1-y)(1-x) + f(0,1)(1-x)y + f(1,0)x(1-y) + f(1,1)xy \qquad (2.17)$$

The formula 2.17 can be derived from a matrix:

$$f(x,y) \approx [1-x \quad x]\begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix}\begin{bmatrix} 1-y \\ y \end{bmatrix} \qquad (2.18)$$

The method of predicting of pixel value is called bi-linear interpolation. Namely it is possible to estimate the pixel information at (x,y) from the present pixel value. The estimated pixel value could improve the efficiency of LBP feature but at the cost of more complex computing.

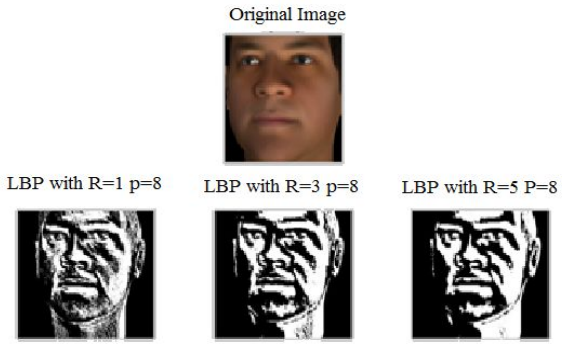

Figure 2.17 *Different textures obtained by changing the LBP radius*

Program 1 below is the practical implementation of LBP calculation in Matlab with results shown in *Figure 2.17*. There are three cases with different radius value and corresponding number of pixels. As can be seen the face detail is reduced with the increasing radius.

```
%initial the parameter for the LBP
R= 5;
P= 8;
img = rgb2gray(imread('3.bmp'));
LBP_img = zeros(size(img));
pad_img = zeros(size(img)+R*2);
% padding the image
for k= 1:R
pad_img(k,1:end-R*2) = img(1,:);
pad_img(end-(k-1),1:end-R*2) = img(end,:);
pad_img(1:end-R*2,k) = img(:,1);
pad_img(1:end-R*2,end-(k-1)) = img(:,end);
end
pad_img(1+R:end-R,1+R:end-R) = img;
% calculate LBP
[row col] = size(pad_img);
%test the radius is 1,3 and 5
if rem(R,2)~= 0
for i= 1:row-2*R
   for j = 1:col-2*R
      center = pad_img(i+R,j+R);
      p0 = sign(center-pad_img(i,j));
      p1 = sign(center-pad_img(i,j+R));
      p2 = sign(center-pad_img(i,j+2*R));
      p3 = sign(center-pad_img(i+R,j+2*R));
      p4 = sign(center-pad_img(i+2*R,j+2*R));
      p5 = sign(center-pad_img(i+R,j+2*R));
      p6 = sign(center-pad_img(i,j+2*R));
      p7 = sign(center-pad_img(i+R,j+R));
      LBP_img(i,j)= p0*1+p1*2+p3*8+p4*16+p5*32+p6*64+p7*128;
   end
end
end

```

Program 1. *Matlab code for LBP with odd radius with 8 patterns.*

From the standard window size of 3×3 the LBP generates 8 bits binary sequence. This means it is possible to produce 256 different binary patterns. It was found that 90% patterns are concentrated in 58 different values according to a huge image statistics analysis [10]. These 58 models are also called uniform patterns. *Table 1* shows the uniform patterns.

Table 1. *58 uniform patterns of LBP*

Index	Binary code	Decimal	Index	Binary code	Decimal
1	00000000	0	30	10000000	128
2	00000001	1	31	10000001	129
3	00000010	2	32	10000011	131
4	00000011	3	33	10000111	135
5	00000100	4	34	10001111	143
6	00000110	6	35	10011111	159
7	00000111	7	36	10111111	191
8	00001000	8	37	11000000	192
9	00001100	12	38	11000001	193
10	00001110	14	39	11000011	195
11	00001111	15	40	11000111	199
12	00010000	16	41	11001111	207
13	00011000	24	42	11011111	223
14	00011100	28	43	11100000	224
15	00011110	30	44	11100001	225
16	00011111	31	45	11100011	227
17	00100000	32	46	11100111	231
18	00110000	48	47	11101111	239
19	00111000	56	48	11110000	240
20	00111100	60	49	11110001	241
21	00111110	62	50	11110011	243
22	00111111	63	51	11110111	247
23	01000000	64	52	11111000	248
24	01100000	96	53	11111001	249
25	01110000	112	54	11111011	251
26	01111000	120	55	11111100	252
27	01111100	124	56	11111101	253
28	01111110	126	57	11111110	254
29	01111111	127	58	11111111	255

The value which is not uniform pattern is classified into 59^{th} binary pattern. With the uniform pattern the number of binary sequence values is reduced from 256 to 59. The reduction into unique patterns considerably accelerates the calculation of LBP features.

2.3.1 Derivation of LBP Invariance

LBP has better properties than other methods to represent the texture of faces due the simplicity of calculation and non parametric characteristics. Besides, LBP performances

is robust in gray level images an for rotation. The following shows the process of the derivation of LBP calculation formula.

1. Invariance to rotation

In the image processing with rotation the property of pixel information would not change. It is apparent that there are 58 uniform patterns in window with size of 3×3. By calculating the LBP map of original image there must occur many repeated binary sequences with varied rotation. These repeated binary sequences are called rotated LBP. They may not belong to identical area in the image but the LBP would be in the relationship of rotation with the others. *Figure 2.18* shows how the LBP value of 15 would would get changing values by rotating it.

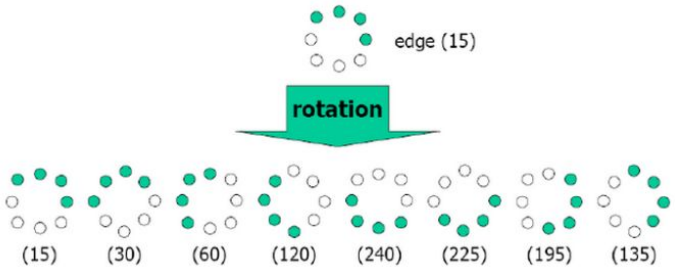

Figure 2.18 *LBP feature which is sensitive to the rotation*

To remove the duplicated rotated LBP the formula 2.15 has to be updated into:

$$LBP_{P,R}^{ri} = \min\left\{ROR\left(LBP_{P,R}, i\right) \mid i = 0,1,\ldots, P-1\right\} \qquad (2.19)$$

ROR(x,i) means shifting the binary value *x* by *i* units along right direction. The LBP feature value becomes independent from the rotation since with the applying of the formula the rotated texture should be normalized into the minimum value as in *Figure 2.19*.

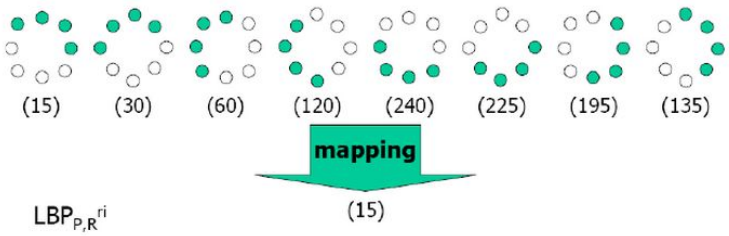

Figure 2.19 *LBP feature with rotation Invariance*

By removing the rotation effect the LBP feature numbers is reduced to 36 from 256 in 8-bit binary sequence. Those 36 features assemble as the basis features of LBP which are illustrated in *Figure 2.20*.

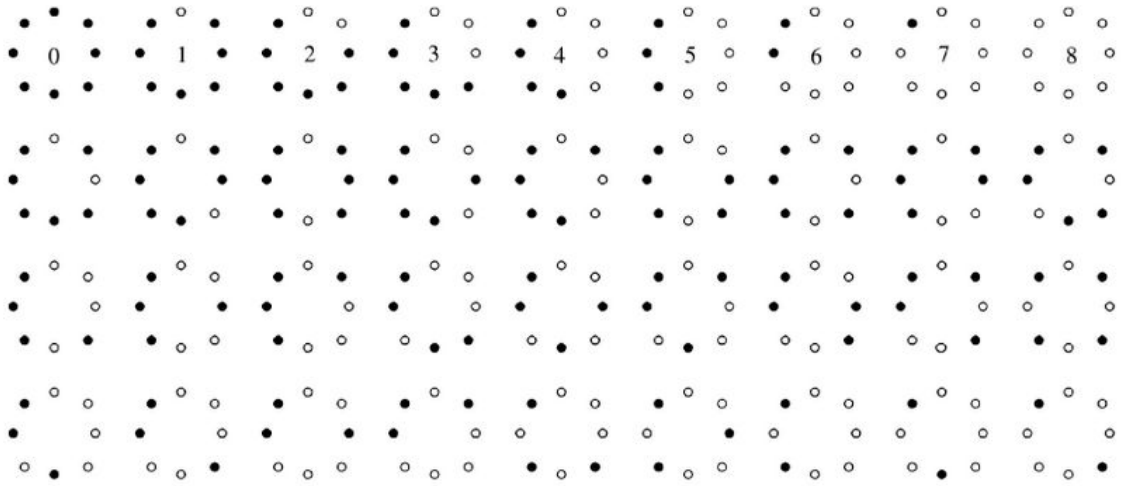

Figure 2.20 *The 36 basis feature values of LBP*

2. Boosted LBP for rotation

Based on the formula 2.19 the rotation effect on the LBP value can be eliminated. The ordinary 59 uniform patterns are reduced to 36 after removing the impact of rotation. However among the 36 features the frequency of each feature is not equal. Methodology for presenting the status of unique feature was defined in [10].

The so-called hop is used to describe the change from 1 to 0 or 0 to 1 in binary sequence values. For example binary sequence 00111111 has one hop but the sequence 11011111 has 2 hops. From the statistics of the 58 features in Table 1 the statement of unique feature can be indirectly described by hop. Most frequently appearing features have the counting of hops smaller than 3. A derived formula is proposed:

$$LBP_{P,R}^{riu2} = \begin{cases} \sum_{P=0}^{P-1} s(i_p - i_c) & if \quad U(LBP_{P,R}) \le 2 \\ P+1 & otherwise \end{cases} \tag{2.20}$$

with

$$U(LBP_{P,R}) = \left|s(i_{p-1} - i_c) - s(i_0 - i_c)\right| + \sum_{P=0}^{p-1} \left|s(i_p - i_c) - s(i_{p-1} - i_c)\right| \tag{2.21}$$

In the formula 2.21 *s(x)* is the sign function with *s(x>0) = 1*and *s(x<1) = -1*. *U(x)* is the condition to filter the different hops. i_c represents the center pixel value and i_p are the surrounding pixels in the image.

2.3.2 Face Representation with the LBP

For the representation with the LBP the face image is first divided into blocks. Next histogram of LBP in each block is used to describe the property of the block area. The complete face image can be described by the concatenation of histograms of all blocks. Mathematically the face representation is thus could described by formula 2.22

$$FeatureofFace = [h1,h2,....,hm] \tag{2.22}$$

where *m* is the number of blocks and *h* is the histogram of each block. For blocks of a face image there is no limitation of block size an and number of blocks. *Figure 2.21* shows different ways to split the face image into blocks. The standard method is to divide images evenly into blocks as in *Figure 2.21(b).* In special cases for example if the eye feature is most significant the blocks could be implemented as *Figure 2.21(c)*. Various types of blocking are acceptable but the most adequate one could supply the best representation.

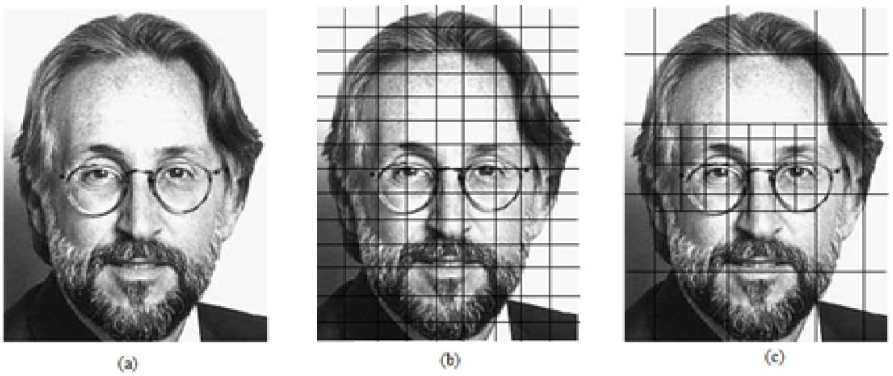

Figure 2.21 *The different blocks of an image (a) original image (b)evenly divided blocks (c) selected divided blocks*

The representation of features varies with the number of blocks. Generally more blocks brings better result for face descriptions since most of details of the face remains. However the large amount of blocks increases the complexity of computing. Smaller blocks can result in redundant features or noise which has no relation to face details. In order to guarantee the speed of calculation and the correction rate of classification suitable blocking should be applied for the face representation. *Figure 2.22* shows the method of concatenating the block histograms.

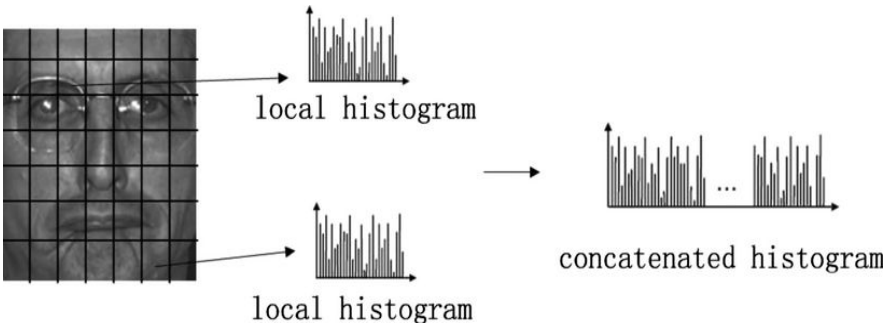

Figure 2.22 *Concatenated histograms as the LBP feature description of face*

2.3.3 Calculation of Similarity for LBP Histograms

To classify data as face or non face one has to compare the detected and training image samples. In the case of LBP one has to compare the concatenated block histograms. Generally there are three methods to measure the similarity of histograms:

1.Chi Square Statistic

$$x^2(h_1, h_2) = \sum_{i=1}^{I} \frac{(h_{1i} - h_{2i})^2}{h_{1i} + h_{2i}} \tag{2.23}$$

2. Histogram Intersection

$$D(h_1, h_2) = \sum_{i=1}^{I} \min(h_{1i}, h_{2i}) \tag{2.24}$$

3.Log-likelihood Statistic

$$L(h_1, h_2) = -\sum_{i=1}^{I} h_{1i} \log h_{2i} \tag{2.25}$$

In the formulas above, h_1 and h_2 are the histogram for both training and detected samples. i is the index of bins in the histogram. Different formulas have various performance in measuring of the similarity of data. Formula 2.23 is using the mean of variance for the discrimination. Log-likelihood method has inferior behavior in handling zero values due to the *log* operation. This means that formula 2.25 behave worse than other two methods if the block size of LBP is relatively small. Chi-Square is considered better than the two others and most often used. classifiers for face detection.

3. CLASSIFIERS FOR FACE DETECTION

In the previous chapter the Haar and LBP features were introduced. These features are used to generate the description for face detection based on machine learning. In this chapter classifiers used for face detection are described. First there is description of the Adaboost method using weak and strong classifiers.

3.1 Adaboost and Gentle Adaboost

Adaboost algorithm was developed by Freund and Schapire in 1996 [8]. Generally Adaboost is a supervised classifier. The training process in Adaboost is produced by allocating weights to data samples. Varying the weight distribution improves the incorrectly classified data samples chance to get correct labels in the next training processing. Adaboost is an iterative algorithm also called discrete Adaboost. Given a training set $(x_1,y_1),..,(x_n,y_n)$, where x_i is the training set sample and y_i is the label in which 1 means face data and 0 is the non face sample, the primary procedure of Adaboost can be stated in 3 steps:

1. Initialize the training session with equal weights. For example if there are N training samples then give every sample the same weight: $1/N$.

2. Classify the samples with local feature as threshold and compare the classification result with original training data label. During the training process the weight of the sample with incorrect classification will be increased in next training processing and the weight of sample with correct detection will be reduced. Allocate the weights in successive trainings.

3. Select the classification result with minimum error. Calculate the error rate for classification. Store false classification error in both positive and negative results. Calculate the threshold for a weak classifier. Save the parameters of local feature and false error to a weak classifier h_1.

4. Repeat t times to generate t weak classifiers $\{h_1,h_2,...h_t\}$.

5. Give the condition to create a strong classifier. Implement the weak classifiers into the processing until the minimum boundary of condition arrives. Save a weak classifier to boosted classifier. Loop the m times by allocating the weights for samples and combine weak classifiers into a boosted classifier H. Boosted classifier also called strong classifier which could effectively remove the noise samples or non face samples.

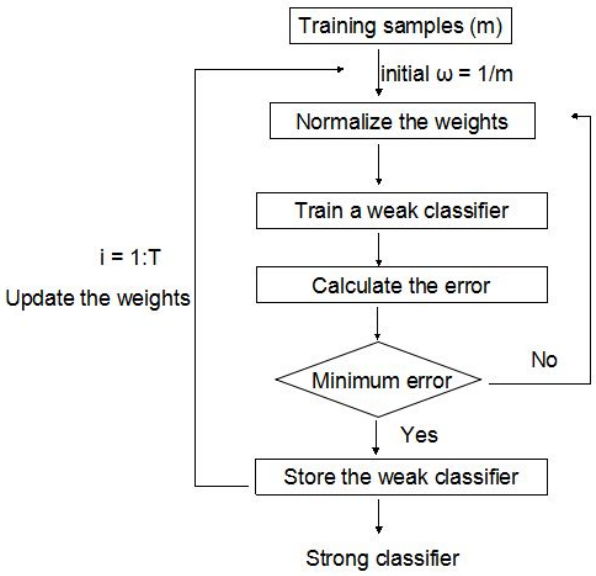

Figure 3.1 *The process of Adaboost*

In *Figure 3.1* the diagram of the Adaboost algorithm is shown. The initial operation in the face detection is to train a weak classifier. A strong classifier contains more than 1 weak classifier. To select the optimal weak classifier the algorithm applies the minimum classification error rate. The error in mathematical form is as in formula 3.1

$$\varepsilon_t = \sum_{i=1}^{N} \omega_{ti} \, | h_t(x_i) - y_i | \tag{3.1}$$

where $h_t(x_i)$ is the classification value for *i-th* sample data by the *t-th* weak classifier and $h_t(x_i) \in \{1\ 0\}$, y_i is the *i-th* sample label and ω_{ti} represents the updated weights. The reallocation of weights in each loop increases the opportunity of for correct classification of the misclassified samples. In Adaboost the update formula for weights is:

$$\omega_{(t+1)i} = \omega_{ti} \beta_t^{1-|h_t(x_i)-y_i|}, i = 1,2,\ldots,N, \beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} \tag{3.2}$$

In formula 3.2 when the supposed feature was incorrectly assigned then $h_t(x_i)$ will be close to 0 if $y_i=1$ and β increases regarding to the large classification error. The exponent of β restricts the scope of weight increasing.

The final strong classifier is as:

$$H(x) = \begin{cases} 1 & \sum_{i=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}, \alpha_t = \log \frac{1}{\beta_t} \tag{3.3}$$

In formula 3.3 α_t is the weight of weak classifier in the strong classifier. One can notice that when $\varepsilon_t \le \frac{1}{2}$, $\alpha_t \ge 0$ and α_t will increase with the decreasing of error. In other words, the weak classifier with lower error rate has dominating influence on the strong classifier.

So far algorithm and theory of basic discrete Adaboost has been described in detail. There are other versions of Adaboost: real Adaboost, Logitboost and gentle Adaboost. Gentle Adaboost was developed by Viola and Jones in 2001 for the improvement on the basic one. Gentle Adaboost was certified [7] as the best performing in the face detection.

Although Gentle Adaboost has the same background as the basic discrete one, slight distinctions between them exist such as the method of weight updating. Different from discrete Adaboost using [0 1] for the representation of samples, the gentle Adaboost employs [-1 1] for the labeling of data. Given the training set $(x_1,y_1),..,(x_n,y_n)$, $x \in \Re^n, y_i \in \{-1,1\}$ the implementation of the algorithm can be concisely described as:

1. Initialize the weight for the training set.

2. Train a weak classifier.

3. Train a strong classifier.

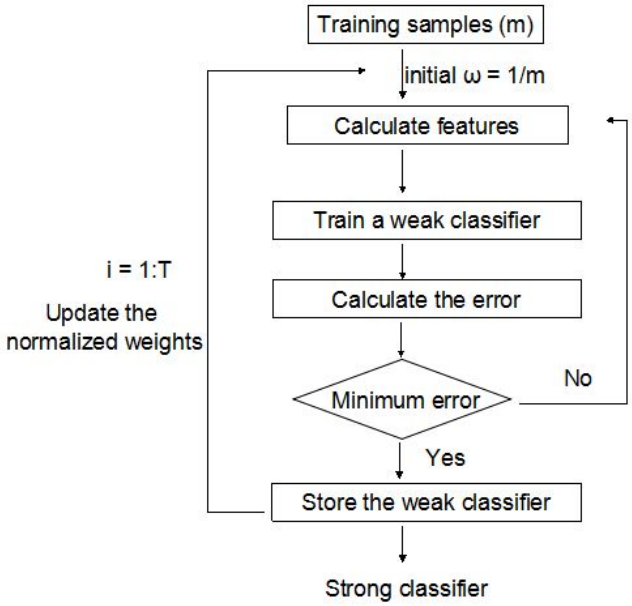

Figure 3.2 *The diagram of gentle Adaboost*

From *Figure 3.2* in the processing of gentle Adaboost differs in the label of features being {-1 1}. The expression for the weak classification is as in formula 3.4

$$h_t = \begin{cases} \alpha 1 & f(x_i) < \theta_t \\ \alpha 2 & otherwise \end{cases} \quad (3.4)$$

where $f(x_i)$ is the feature value and θ denotes the threshold for the classification, α describes the classification result and $\alpha \in [-1,1]$. Since the label and weak classifier have different representations, the weight renewing for gentle Adaboost is as [6]:

$$\omega_i \leftarrow \omega_i \exp[-y_i \cdot h_t(x_i)] \qquad i = 1,2,...,N \quad (3.5)$$

From the formula (3.5) the updated weights need to be normalized. The strong classifier in gentle module is then:

$$H(x) = sign(\sum_{t=1}^{T} h_t(x)) \quad (3.6)$$

where *sign* is the comparison operation in which the sign(x) = 1 when $x \geq 0$ otherwise sign(x) =-1. The strong classifier combines directly the weak classification result whereas in discrete Adaboost the weights were assigned into the weak detection value.

3.2 Structure of the Classifiers

The structure of the classifiers is depicted in *Figure 3.3* for the training and detection parts. Classifier is composed of several parts. After the Haar or LBP feature calculation there are several blocks which are described as weak, strong and cascaded classifiers.

Weak classifier is the basis of classifier for both strong and cascading mode which uses the local feature value to filter out the best feature. The result of the weak classifier is the possibility of the classification which means the detection result just represents the correct classifying percentage. Differently from weak classifier the strong classifier directly gives the result as 1 representing the face and 0 for non face. Strong classifier generally is made by the concatenation of the weak classifiers. Cascading classifier is a higher level classifier which includes a number of strong classifiers in different stages. Each stage of cascading will filter out the non face sub-windows or suspected non-face samples and leave the samples which are the most likely to be the faces.

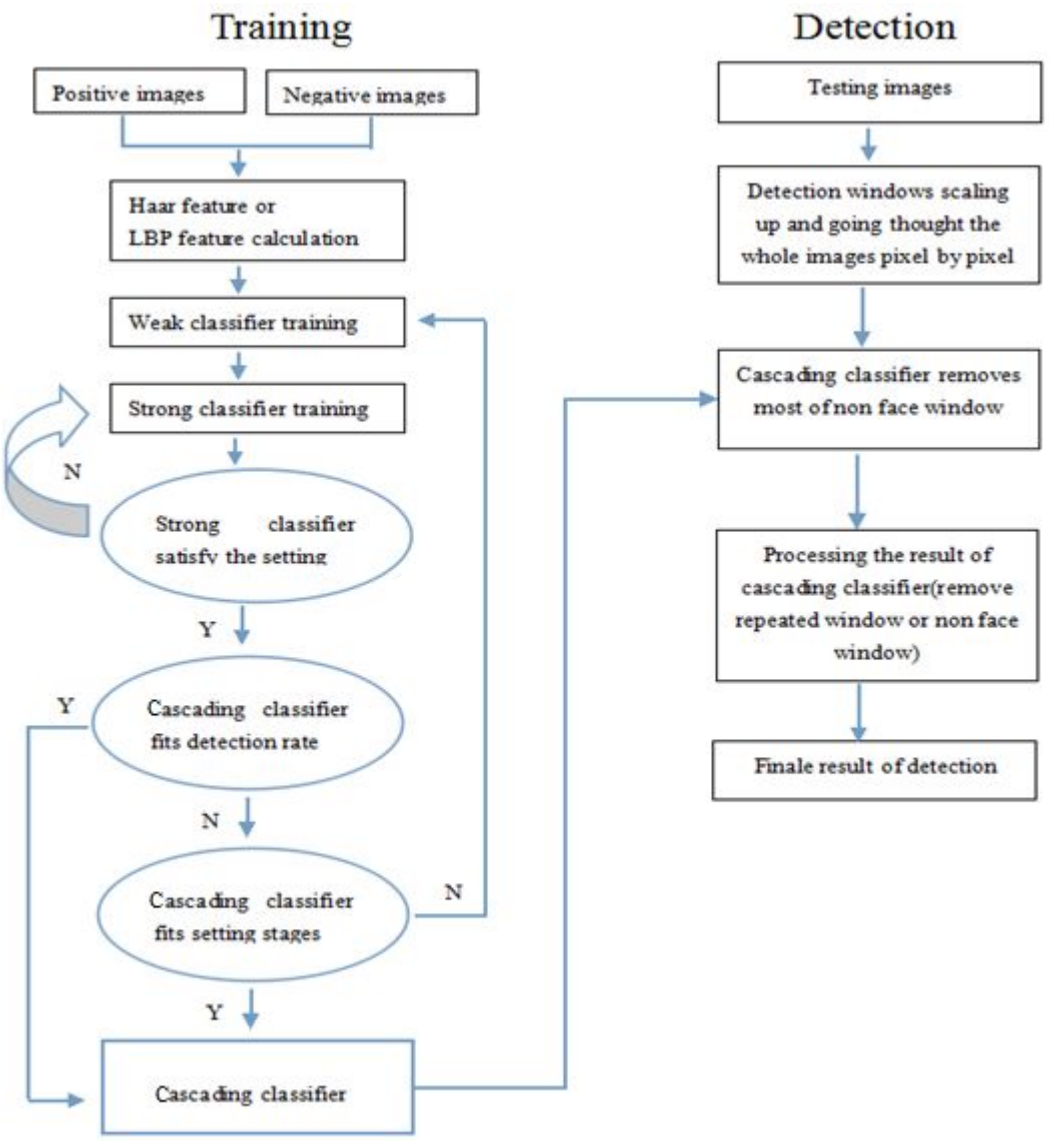

Figure 3.3 *The structure of classifiers*

3.3 Classifier Training

In this sub-chapter we will focus on the weak and strong classifier training algorithm and briefly introduce the operation of cascading classifier.

Typically the core of training of the weak classifier is to find appropriate threshold which could supply minimum classification error. Strong classifier is based on a number of weak classifier in order to guarantee best classification results.

3.3.1 Weak Classifier Training

Weak classifier training is the basic and essential procedure in the face detection process. Basically the weak classifier training can be summarized as follows:

1. Prepare the training samples and the number of Haar features. Calculate the features for all samples.

2. Save the feature values to a matrix.

3. For each feature in the matrix, classify the training samples by comparison with remaining features successively. Assign the sample as face if the current feature is bigger.

4. Calculate the error and save the parameters of feature with minimum error.

Figure 3.4 shows the detail of weak classifier training. Suppose there are *m* samples and *n* Haar features, for each feature we could obtain *m* values for all samples. These *m* values of feature are sorted by increment to reduce the complexity of classification. For example given current feature is at *i-th* location of feature matrix then the left *i* features were assigned as face and right *m-i* features were seen as non face. The classification establishes thus left and right partial classification. The classification error of left and right can be calculated as in formula 3.7 and 3.8

$$lefterror = \sum_{k=1}^{j} \omega_k \cdot (y_k - leftvalue)^2 \tag{3.7}$$

$$righterror = \sum_{k=j+1}^{m} \omega_k \cdot (y_k - rightvalue)^2 \tag{3.8}$$

In the formula ω and y is respectively the weight and label of training sample, *leftvalue* and *rightvalue* represent the grouping of the values which is defined as:

$$leftvalue = \sum_{k=1}^{j} \frac{(\omega_k \cdot y_k)}{\sum_{k=1}^{j} \omega_k} \tag{3.9}$$

$$rightvalue = \sum_{k=j+1}^{m} \frac{(\omega_k \cdot y_k)}{\sum_{k=j+1}^{m} \omega_k} \tag{3.10}$$

The error of classification is *error* = *lefterror* + *righterror*. The training process attempts to obtain the feature which generates the minimum error. Once the system catches such feature the training stores the parameters of this feature into a new weak classifier.

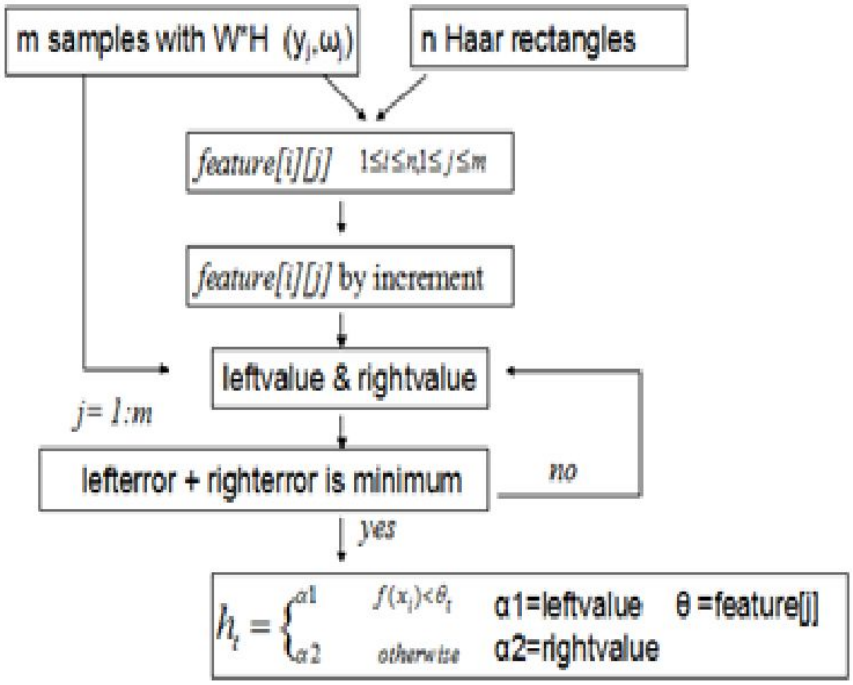

Figure 3.4 *The processing of weak classifier*

3.3.2 Summary of Weak Classifier Training

Weak classifier training is the adaptive learning process for a single feature. The feature classification is implemented by the determination of value of threshold. Every feature can create a weak classifier by classifying the samples with the local feature value. Threshold value in a weak classifier points to the feature value of samples. The features of all samples calculated by all Haar feature patterns are saved as a feature matrix. Each row of the feature matrix represents the value of all samples regarding to a Haar feature. By converting the entire row into an ordered vector which is called feature vector, the vector can be applied to classify the ordinary samples by local values and calculating the left and right errors separately. The classification with minimum error is selected to create a weak classifier. Generally the classification error boundary is limited to 50%. If the value of error exceeds the limitation the classifier training is over-trained or failed. Gentle Adaboost is optimized for the single Haar-like feature and improves the functionality of weak classifier.

The classifier training severely prevents the duplicated features which would result in over-training or dead loop during the training. The reason for over-training is that the unique feature will arise repeatedly but the training system tries to search for the different feature, without the updating of weights the training processing will not stop or will be over-trained. Updating weights is somehow able to regulate the distribution of positive and negative samples. The training will break down if the ratio of positive and

negative tends to 0. With the weight updating the training system will automatically adjust for the number of positive and background images.

For feature calculation the Haar feature uses integral image to achieve rapid computing. Normally the upright features are enough to train a classifier with good classification result. The rotated features in the training absolutely improve the performance of classification but at much higher cost of computations and memory consumption.

3.3.3 Strong Classifier

Strong classifier is defined as the combination of weak classifiers. Weak classifier is optimized for the feature to meet the boundary of false error rate. Initial requirements are employed during the strong classifier training to guarantee an acceptable false error rate.

Supposed the samples are $(x_1,y_1),...,(x_m,y_m)$ in which x represents the training data and y is the label of classification. The number of face samples is np and the amount of non faces is nb with $np+nb=m$. Preliminary weight for positive and negative samples are:

$$\omega_i = \frac{1}{2 \cdot np} \text{ and } \omega_i = \frac{1}{2 \cdot nb} \tag{3.11}$$

Initialize the minimum value of detection rate d_{min} and the maximum false rate f_{max}. Similarly as for the weak classifier calculate the feature value with n Haar-like features for all samples and save the values to a feature matrix.

Let d and f are the current detection rate and false rate with initial d=0 and f = 1. The number of weak classifier in the strong classifier is $t=1$ at beginning. The t-th weak classifier which was selected to be element of strong classifier represents as $h_t(x)$.

Under the condition of $f > f_{max}$ select a Haar like weak classifier $h_t(x)$ with the minimum classification error. The error of weak classification was defined in the formula 3.1. Classify the positive and negative samples independently by current strong classifier and calculate the classification result pos for face samples and neg for non face images in the formula 3.12

$$result[i] = \sum_{j=1}^{t} h_j(x_i) \tag{3.12}$$

where $h_j(x_i)$ is the classification result for i-th sample by j-th weak classifier. To meet the condition of $d \geq d_{min}$ the threshold φ_t is required for current strong classifier

$$\varphi_t = Pos\_sort(np \cdot (1 - d_{\min})) \tag{3.13}$$

In formula 3.13 *Pos_sort* is the sorted vector of *pos*. In order to satisfy $d \geq d_{min}$ at least $np(1-d_{min})$ features should be classified correctly. Set *numoferror = 0* as the false error number in negative images. When the value in *neg* vector meets $neg[index] > \varphi_t$ increase the *numoferror* by 1. Update the current false rate *f* as:

$$f = numoferror/nb \tag{3.14}$$

If $f < f_{max}$ the training finishes otherwise the training continue by updating the weights of samples like formula 3.5, next renew the number of weak classifier by increment. The entire training process can be illustrated as in *Figure 3.5*.

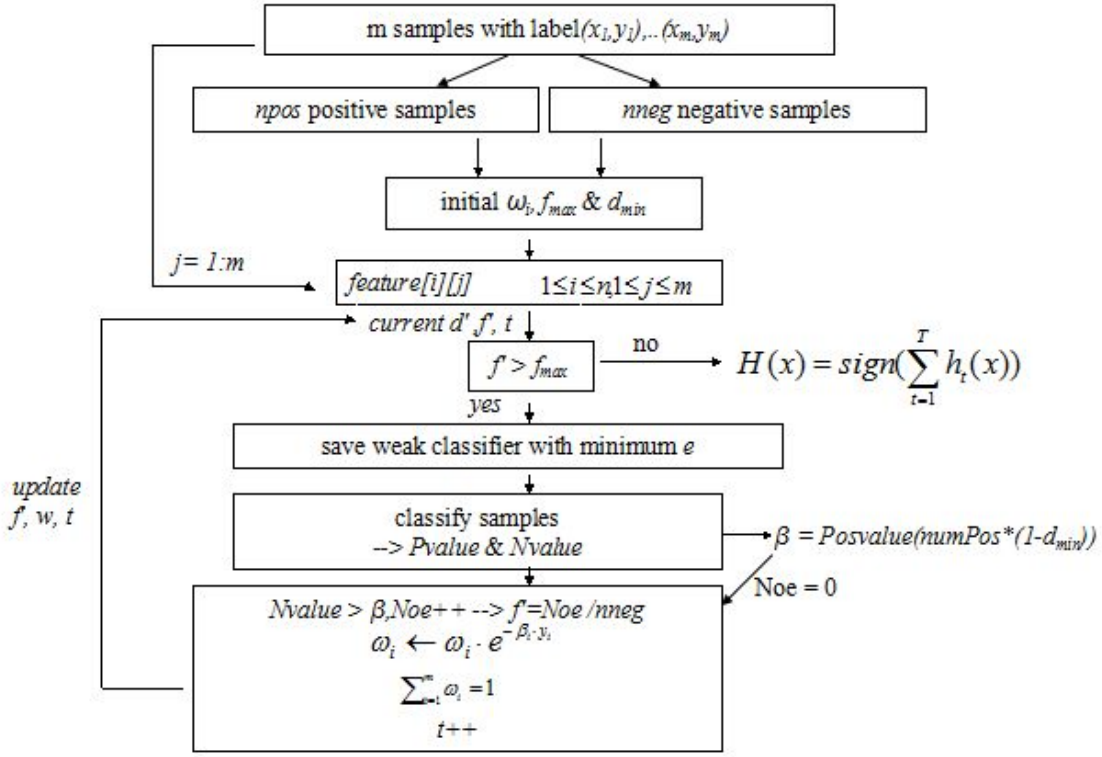

Figure 3.5 *The procedure of strong classifier training*

3.3.4 Cascading Classifier

Cascading classifier is the arrangement of series of strong classifiers. The performance of detection primarily is determined by the stage of cascading. With cascading classifier the training behaves the high detection rate and low false error rate. The connection of false error rate and detection rate regarding to the number of stage is outlined as:

$$Detection\ rate = minhitrate^{stage} \tag{3.15}$$

$$False\ error\ rate = maxfalsealarm^{stage} \tag{3.16}$$

minhitrate represents the minimum rate of the positive detection and *maxfalsealarm* is the maximum detection rate for negative images. In practical training the *minhitrate* and *maxfalsealarm* assign the values of 0.998 and 0.50. For instance the cascading classifier with 20 stages proves the relatively high detection rate of 96% and extremely low false error rate of 0.000095%.

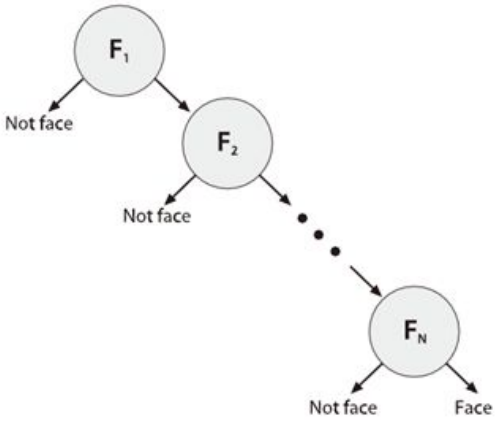

Figure 3.6 *The diagram of a cascading classifier*

The principle of cascading classifier is depicted in the *Figure 3.6*. The classifier contains *N* stages with the specific detection rate *d* and false error rate *f* at each stage. The non-face data is removed gradually by every step of classifier. The sample carrying out through all of the stages is accepted as the ultimate face data. Generally the most of non-face samples are removed in the preliminary stages, in other words the beginning stages of cascading have major role on detecting the negative data and the remaining stages take on the face detection.

Let the correct rate for *i-th* stage of cascading is p_i where *i* is the index of cascading, n_i is the number of features in *i-th* stage of cascading. The average number of features employed in *i*-stage of cascading classifier is:

$$n' = n_k + \sum_{k=1}^{i} (n_k \prod_{j<k} p_j) \tag{3.17}$$

From this formula it is obviously $n' \leq \sum_{k=1}^{i} n_k$; it means that cascading classifier will need less features than a strong classifier. Meanwhile the response to detection behaves better than with the strong classifier.

Suppose the maximum of error for the cascading classifier is F_{max} , minimum detection rate and maximum error on each stage are respectively d_{min} and f_{max}. The number of stages of cascading will be then as in formula 3.18

$$M = \frac{\log(F_{\max})}{\log(f_{\max})} \tag{3.18}$$

Given the number of positive samples is *NP*, and the positive training set is *numPos* with *numPos* < *NP*. The number of negative samples is *NN* and the training non face samples number is *numNeg* with *numNeg* < *NN*. Calculate the feature for both positive and negative samples and save the values in a feature matrix. The ratio of positive and negative expresses as *ratio=numNeg/numPos* and the initial false error for cascading classifier is $F_0 = 1$.

At the *i-th* stage train strong classifiers with $f_i \le f_{\max}$ and $d_i \ge d_{\min}$ and cascade the current *i* stages. Update the false error rate of classifier as $F_i = f_i \cdot F_{i-1}$ if $F_i < F_{max}$ the training processing is completed otherwise classify the positive and negative training set with the current cascading classifier. Afterwards *n* face samples are successfully detected and *k* non-face images are found. Renew the positive training number $numPos \leftarrow n$ and replace these *k* detected training samples with the remaining of negative images where $numNeg \leftarrow \frac{numPos}{ratio}$.

In *Figure 3.7* the chart of cascading classification process is shown, the framework of cascading classifier and the break condition for the training

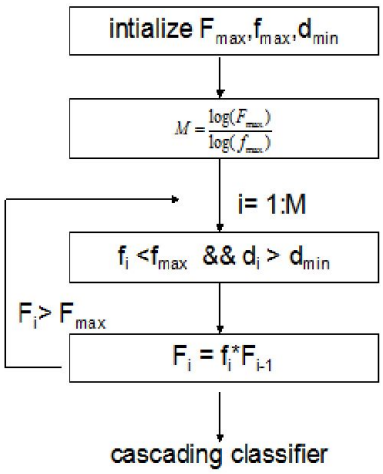

Figure 3.7 *The diagram of cascading classifier*

In the training of cascading classifier there are some problems worth to pay close attention. The overtraining may happen during the training. It is necessary to fix the ratio of positive and negative training samples number to a proper value which has the capacity of avoiding the over-training or dead loops. Also updating the samples per stage helps efficiently to solve this problem. Due to the considerable low false error of

cascading classifier, the influence of positive data deletion in the process of misclassification will be small.

3.4 Fixed Detection Window

The goal of the detection processing is to acquire the face-similar area by scanning the entire image. The image scanning can be done with two different methods: fixed detection windows and scaled detection windows. In the former the searching window has constant size as the training window and in the latter keeps the image size constant. When having the detection window with fixed size we have to repeatedly scale down the original image to suitable size in order to detect the face correctly. The image is scaled down by fixed ratio until its size and the size of window are approximately similar as shown in *Figure 3.8*. To detect the human face by fixed detection window should emphasize the scaling coefficient of the original image. The smaller scaling may result in many redundant non-face areas and the larger scaling has the problem of missing faces. Scaling down the original image may also change the pixel information or contrast of the original image.

Figure 3.8 *Fixed detection window and scaling down image*

In addition the integral image has to be recalculated when the original image is re-sized. Normally the detection processing with fixed window is not proposed in practical implementation.

3.5 Scaled Detection Window

Different from fixed detection widow the scaled detection window maintains the size of image and varies the size of searching window. The identical image size avoids the re-computing of integral image but the varied searching windows result in repeated calculation of features. Compared with the calculation of integral image the feature calculation is relatively simpler.

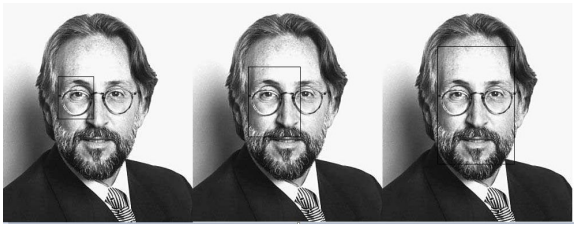

Figure 3.9 *The fixed image size and variable size of detection window*

Feature invariant to the change of window size can be derived starting from the formula (2.2). Given the window size of dimensions *w* and *h*, the area of the window is $Area = w \cdot h$, the boosted Haar like feature can be expressed as follows:

$$feature = \frac{feature_{old}}{\sigma \cdot Area} \tag{3.19}$$

$$\sigma = \sqrt{\frac{\sum_{x=1}^{w}\sum_{y=1}^{h} I_{(x,y)}^2}{Area} - \left(\frac{\sum_{x=1}^{w}\sum_{y=1}^{h} I_{(x,y)}}{Area}\right)^2} \tag{3.20}$$

$I_{(x,y)}$ is the pixel value at *(x,y)* in the window and σ is lighting normalization factor to remove the influence of light.

3.6 Processing of the Detection Windows

After detection the face area in an image will be outlined by a box depending on the scale of the detection window. There would be then several redundant windows which would be treated as misclassification of the face. *Figure 3.10* depicts the situation with and without removing redundant windows, one can see multiple windows with face detected.

Figure 3.10 *The processing of redundant detection windows*

Except special condition that the cascading classifier is perfect that only faces are detected, there are also incorrect detection windows with no faces and these windows have to be eliminated. The duplicated windows might have a particularly unique or completely different size. Examples of repeated windows are illustrated in *Figure 3.11*.

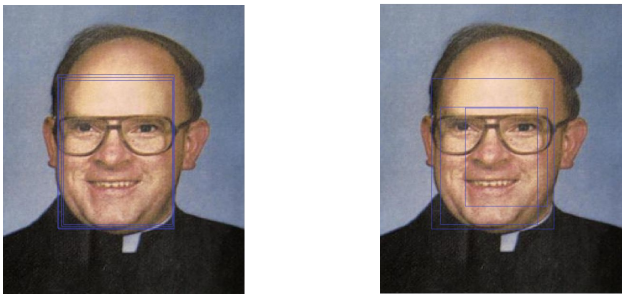

Figure 3.11 *Different cases of duplicated windows*

The principle of deleting the misclassified and repeated windows is to measure the dimension of the overlapping area. Generally if the size of overlap is bigger than the half of current window size then both windows are combined into one. There are essentially two steps to remove the redundant windows:

1.Removal of overlapping windows

Suppose that there are two different rectangle windows denoted as $window_1=(x_1,y_1,w_1,h_1)$ and $window_2=(x_2,y_2,w_2,h_2)$. The threshold of overlapping area can be set as θ which default value of 0.5. When with the threshold the windows satisfy the condition $x_1 \le x_2 \le x_1 + w_1 \cdot \theta$, $y_1 \le y_2 \le y_1 + h_1 \cdot \theta$ and $\frac{w_1}{2} \le w_2 \le w_1$ then the two windows are combined into one. The coordinate of new window will be assigned as the mean of parameters of the two duplicated windows.

2. Removal of non-face windows

Generally the windows of incorrect classification is different from face windows. The cascading classifier supplies extreme low false error rate so there is little possibility of happening that multiple windows overlapped in non-face area. It is natural that the face area will have more overlapping windows than incorrect detection area. So we can use the amount of duplicated window to remove the non-face windows. Given a threshold of minimum allowed repeated windows *min_window_overlap,* the value of this threshold is empirically set to 1 or 2. If the threshold is relatively large then some faces with special pose will be recognized as non-face and removed. Practically threshold value of 2 can efficiently remove most of irrelevant windows.

3.7 Summary of the Face Detection Process

Currently there are different methods of face detection in images. Using machine learning to train a classifier is one of the popular methods in practical applications. In the thesis the training system based on Haar and LBP features is used to produce a robust classifier. Weak classifier is used as the basic and important element for the cascading classifier. The core of the weak classifier training is to search for an

acceptable threshold for the detection with the minimum error. Weak classifiers can be grouped into a strong classifier Varied arrangements of weak classifiers could be grouped as a strong classifier. To optimize the performance of detection for strong classifier the minimum false error rate and maximum detection rate are used in the training processing. Similar as with the strong classifier the cascading classifier is a combination of strong classifiers.

4. EXPERIMENTAL ENVIRONMENT AND PERFORMANCE EVALUATION

4.1 The Data Sets

The face detection training and testing in the thesis was implemented on personal computer. Personal computer performance is acceptable for the medium number of training samples. Two types of positive image were utilized during the training: images with real human face and images with face generated by professional 3D graphics software. The real human face data was freely downloaded from the Face Detection Data Set and Benchmark database (FDDB) [11] and 3D graphics samples were generated by a software called FaceGen. Negative samples without face information were sourced from Personal Event Collection data set (PEC) [12]. *Figure 4.1* displays the samples from FDDB and the negative samples are illustrated in *Figure 4.2*.

Figure 4.1 *Real human face samples from the FDDB*

FDDB primarily focuses on the frontal human face with variable direction, emotion and intensity. The diversity of faces benefit for improving the performance of training. The negative images are the images completely without human faces data, the PEC supplies image topics with landscapes or nature plants/animals. Due to the non-sensitive to gray level by feature learning classifier the color information of the images is not important.

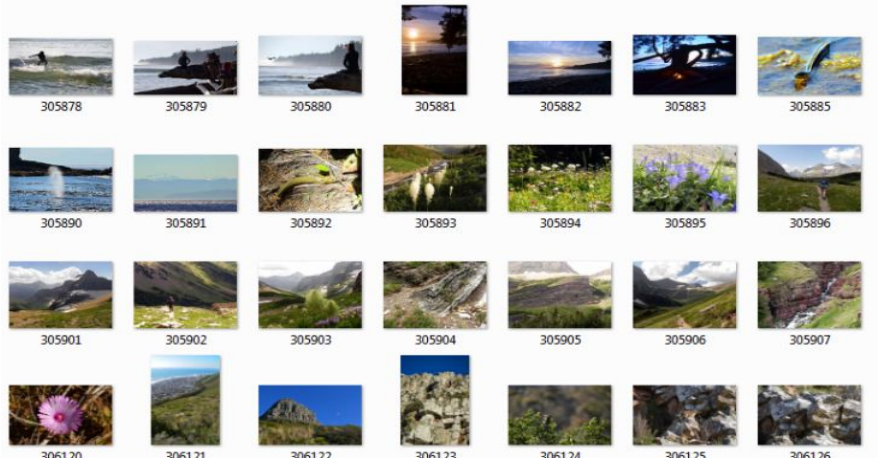

Figure 4.2 *The example of negative images from PEC*

4.2 Introduction of FaceGen Software

FaceGen software is a free and convenient tool to generate 3D human faces. The software contains more than thousands of parameters to modify the model of face. The interface of FaceGen software is as shown in *Figure 4.3*.

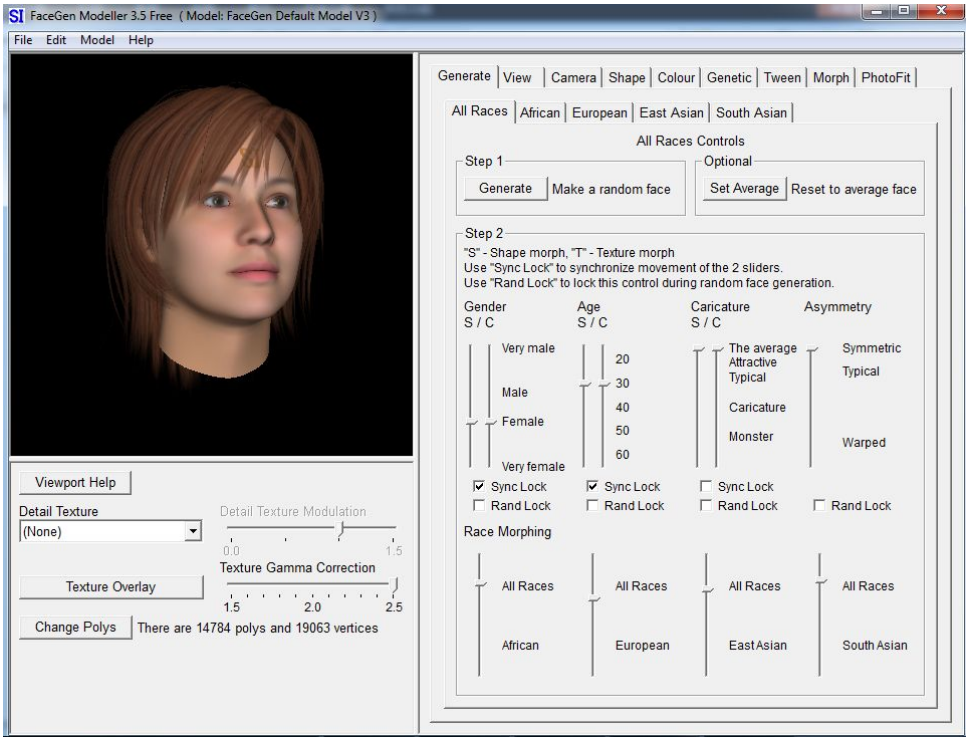

Figure 4.3 *Interface of FaceGen*

FaceGen supports a large amount of functions of which the function view and morph were of interest since our main application using FaceGen includes the random generation of face and its parameters modification. FaceGen can deal with only one face

at a time which is not suitable for our goal since we need thousands of faces. However we were able to deal with this problem by automating the entire processing using software which is able to memorize and record the movement of mouse and keyboard. The details of FaceGen functionalities are introduced next.

4.2.1 The View Function of FaceGen

Regarding the face diversity for classifier training the view class of FaceGen provides changing lighting conditions. This is important since different light intensities produce distinct feature values for an otherwise identical image. Sufficient illumination for training samples is necessary but there are many possible illumination levels.

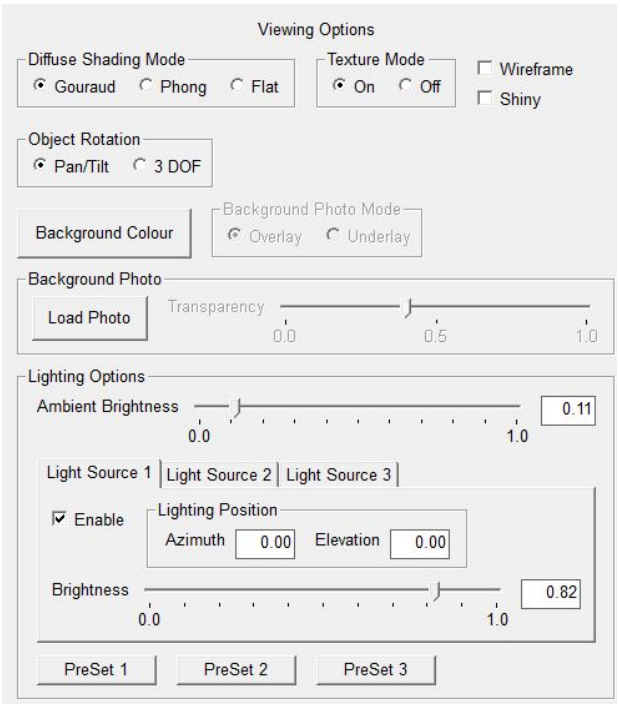

Figure 4.4 *View of the FaceGen interface*

The interface and the main parameters of view of FaceGen are shown in *Figure 4.4*. The background of image is not essential but the performance of classification will be better with different backgrounds. FaceGen viewing supports two types of attaching the background: default background images and customized background images. The critical function in view is the lighting options in which the direction and the intensity of lighting could be altered.

4.2.2 Camera in FaceGen

Classifier training by the method of machine learning requires diversity of training samples in data sets. In order to detect the face with random pose in the photo the

training images should consist of huge data set with all possible poses. The camera class in FaceGen provides good functionality to generate various fields of view and directions.

Figure 4.5 indicates the content of camera function. There are four options to create different poses of face. By changing the values of them the data samples can be produced having numerous poses.

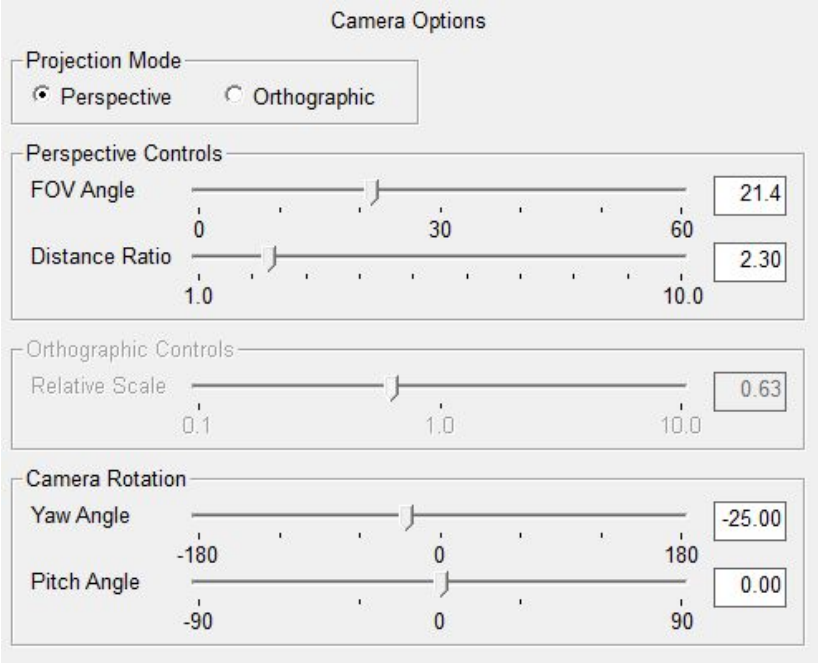

Figure 4.5 *Details of camera options*

4.2.3 Morph in FaceGen

Morphing the face corresponds directly to the expression of emotion of human beings. FaceGen provides 39 face expressions for human face. *Figure 4.6* provides two examples of human face without and with morph or expression.

Figure 4.6 *The example of morph*

It is apparent that samples with morph display more details in faces. The procedure of the graphic face generation used by us with FaceGen is shown in *Figure 4.7*. We used an API in our project named AutoHotKey which is able to record the movements of mouse and keyboard afterward saving them into script code. By randomizing the

parameters in the script code we obtain automatic generation of face data set produced by 3D graphics.

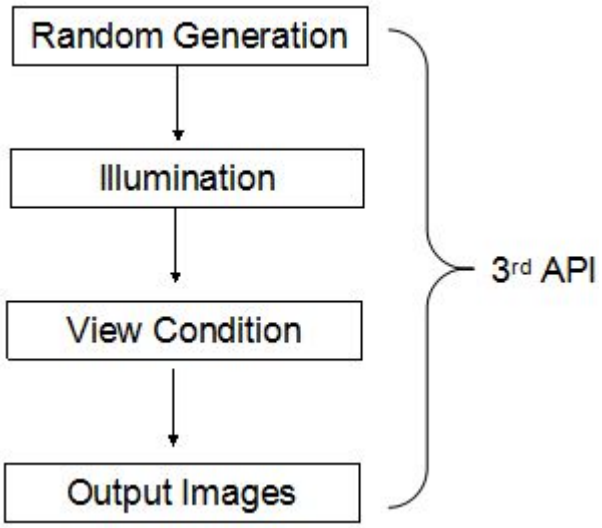

Figure 4.7 *Chart of sample creation with FaceGen*

The final training samples created by FaceGen are shown in *Figure 4.8*. The random generation function in FaceGen behaves with certain bias with respect to gender and race. Among all samples there far more European and male faces than Asian and female faces.

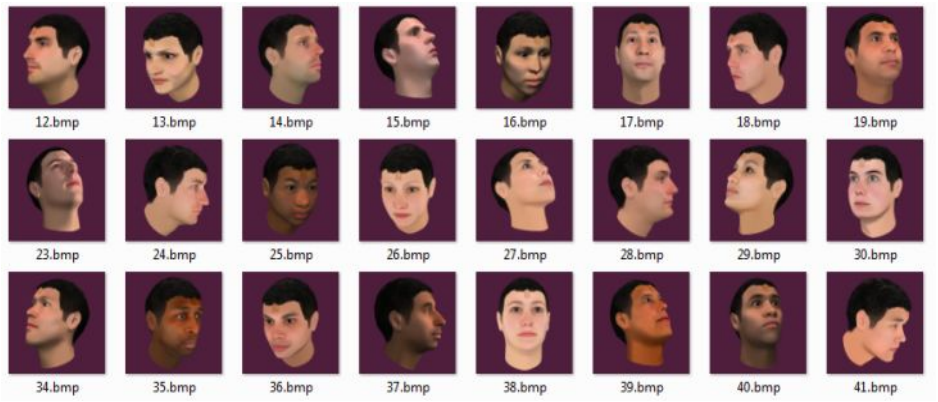

Figure 4.8 *Examples of 3D faces from Facegen*

4.3 Training Processing

The primary tool we used in the training and face detection experiments is the OpenCV which is public C++ library for image processing. Experimental procedures were divided into 3 steps:

1. Preparing the images for positive samples.

2. Training the classifiers with training images.

3. Testing the classifiers and comparing the detection results.

4.3.1 Creation of Positive Images

Positive image creating is the first and necessary step for the training cascade by OpenCV. There is a tool called *Objectmarker* which was published by FuDan University in China. Using the *Objectmarker* it is possible to manually extract the area of human face from samples. The description of rectangle which covers the area of face can be defined as *[i x y w h]* in which *i* is the number of face in the image, *x* and *y* represent the starting coordinates of rectangle and *w*, *h* is the width and height of rectangle.

Rawdata/P1.bmp 2 105 57 97 115 306 123 83 99

Figure 4.9 *The example of face expression by objectmarker*

In *Figure 4.9* "*Rawdata/P1.bmp*" is the directory and name of image and the value of 2 reveals the number of faces in image. The successive numbers represent the dimension and the position of marked windows.

With gathered position information of face, the positive image can be created by the function *createsample* in OpenCV. The file with suffix '*.vec*' expresses the face images. The text file stores the coordinate data marked by *Objectmarker*. The number in the command line implies the number of face images which is not equal to the number of images. For instance with 3000 images one could create more than 3000 face images since more than one face could be in a single image.

The typical images of faces obtained are shown in *Figure 4.10*.

Figure 4.10 *Example of face images (upside real faces, downside 3d faces)*

During the creation of image, OpenCV automatically added noise to the images which may increase the feature diversity.

4.3.2 Training of Classifiers

In our experiments, we prepared both real face and 3D face data sets. The real face image was built from 3800 face images covering more than 10 different directions and variable illumination conditions. 3D face image was created from 3000 images with different positions and small amount of noise was added to them to make them less 'clean'. The training was executed in several groups based on classifiers using the LBP and Haar features. Diversity of data sets is important in the training. If the data samples have only unique pose or contain similar faces there will be over-training. Also the number of positive and negative samples must be selected suitably so that the classifier will have enough training capacity for both positive and negative samples.

Experiments were made with both transform features of Haar and LBP. It is evident that LBP has definitely faster processing, on the other hand Haar produces better results. The target parameters of training such as *minHitRate* and *maxFalseAlarm* determine the performance of training, *minHitRate* defines the minimum correct detection rate and *maxFalseAlarm* reveals the acceptable dis-classified error rate. In practice the value of *minHitRate* is set as 0.998 namely the system will allow two images incorrectly classified in 1000 positive samples. It is a bad case if the number of training samples is too small. It means small number of data samples with high *minHitRate* will result in over-training, to solve this the amount of samples should be increased.

During the training real face data and 3D face data behaved distinctly. At each stage of cascading training many different features were calculated with real face processing while less features were found during the 3D face training.

In the experiments the classifier was trained with more than 3000 real face samples, the number of stages ranged from 19 to 25. For the 3D face training the number of stages was set to 17 due to less detail of features in 3D faces. After completing the training the

classifier saves the results in the target folder as '*cascade.xml*'. The result of processing of cascaded training is illustrated in *Figure 4.11*.

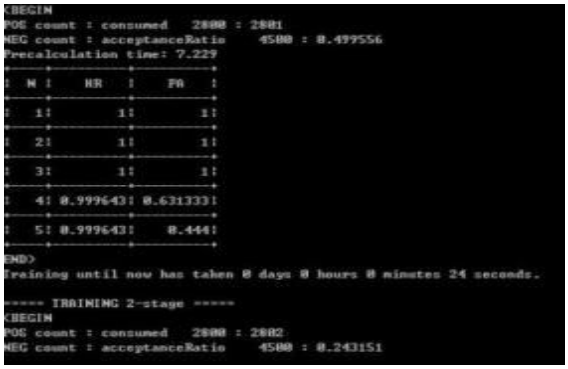

Figure 4.11 *Screen shot of the cascading training*

4.4 Testing Classification

The testing data set is different from the training set. In the experiments we used maximum 3800 real and 3D face images to train a classifier and another different 1000 images to test classifier.

During the face detection experiments the classifiers trained on real faces were tested with the detection of both real and 3D faces from test data sets. Also the classifier trained on 3D face data was tested for real face detection. In the detection process, after marking the detected face region with blue rectangle the processed image is saved into a new folder. By counting the number of correctly and incorrectly classified regions we can calculate the *correct rate = correct classification/ total number* . *Figure 4.12* shows examples of different situations happening in real detection results.

Figure 4.12*. Examples of results of detection*

There are four different cases occurring in the detection. The first is correctly detected human face without any mistakes, the second is the face that has been found correctly but together with wrong objects marked in the image, the third situation is incorrect classification and the last one finds nothing in the image. It is possible to improve the on the second case of detection by increasing the number of negative samples and/or adding more stages for cascading classifier training. The reason for non-detection may

be due to the pose of face which was not included in the training samples or the number of similar samples is not enough.

4.5 Results of Experiments

The experiment was sorted into three parts: real face training and testing, 3D generated faces training and testing and mixed set faces training and testing. Table 2 shows the parameters for experiments and experimental results. In the experiments, we successfully achieved the correct detection rate of 92% from 1000 test images from real face training. The other target was to test the detection rate of 3D faces. For the training with 2990 3D faces, the best correct detection rate achieved was 65%. The results for the 3D faces was not as good as real faces, it may be because of more similarity between the 3D generated faces than in real faces.

Table 2. *The result of experiments*

Training 3d face sample = T3D

Training real face sample = TRF

T3D	TRF	Negative	Stage	Test	Mis-detect	Non-detect	Correct	Ratio
0	3300	4000	25	1000	10	70	920	92%
0	3300	1800	25	1000	328	9	663	66.3%
0	3300	10000	18	100	40	20	30	30%
300	3000	4000	23	1000	19	90	891	89.1%
990	2010	4000	21	1000	13	124	863	86.3%
2990	0	3500	17	1000	48	300	652	65.2%

From the result of the detection, we can see that the real faces provide better performance than 3D faces. There are some possible reasons for the worse result for 3D faces:

1. The 3D face texture is quite smooth, namely the face lacks the details or the real face has more detail than the 3D face.

2. Pose of generated 3D faces are not sufficiently random. The software use does not allow to generate faces with quite different poses automatically.

3. The background of 3D face pictures is blank. It was hard to add different backgrounds automatically to 3D faces. The background does not impact the result much but with the variable background the performance of detection could be better.

4. Illumination of 3d faces is not natural and harmonious like the random lighting situation in real faces. The lighting condition determines the result of detection.

However if there is a software which can generates the faces with wider range of different parameters, the capacity of classifier trained on 3D face data should be identical as for the real face data.

The results above were produced with classifier using LBP features. Classifier based on the Haar features was also tested. Due to the long training time of this classifier two groups of data were used for training with the same computer configuration. Results presented in Table 3 confirm that the detection performance with the Haar features is better than with LBP features but complexity is much higher.

Table 3. *Detection performance and time consumption by Haar and LBP classifiers*

	Samples(p/n)	Test	Time	Ratio
Haar	2990(3D)/3500	1000	About 1 day	71%
	850(real)/1200	100	About half day	78%
LBP	2990(3D)/3500	1000	1 hour 10 min.	65.2%
	850(real)/1200	100	15 min.	70%

The above Haar feature classifier was made with the basic model. The configuration of computer is AMD Athlon II×2 245 Processor 2.90GHz, 6GB memory and 64bit operating system. With full model the detection results will naturally improve but even more time will be consumed for training. The decision which classifier is better should thus include results and computing time.

4.6 Result of Real Time Face Detection

Real time face detection is the practical application of classifier in the real world. After the time consuming training, classifier is able for very fast detection which can be tested with real time face detection. The interaction of classifier and camera determines the quality of detection. Real time detection requires fast computing of features and rapid classification. Due to this the method of calculating the features is important during the training and detection. The experiment with real time face detection proved that the

classifier based on Haar/LBP features performed well. Detection response was practically real time. *Figure 4.13* displays the result of face detection with own computer camera.

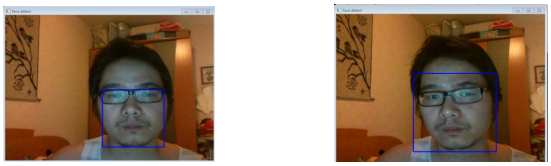

Figure 4.13 *(left) 3D face trained classifier(left) Real face trained classifier*

It is obvious that the detection area depends on the training data, in our case the 3D training face data does not cover the forehead and the detection windows do not cover the forehead too . Program 2 is OpenCV face detection code for PC with camera.

```
#include "opencv2/opencv.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <sstream>
#include <iostream>
#include <stdio.h>
using namespace cv;
using namespace std;
void detectAndDisplay(Mat );
CascadeClassifier face_cascade_l;
String face_cascade_lbp = "./cascade/cascade_p_3k_n_4d5k_s_25.xml";
int main(int, char**)
{
  if( !face_cascade_l.load( face_cascade_lbp ) ){ printf("--(!)Error loading\n"); return -1; };
  VideoCapture capture( 0 );
  for(;;){
    Mat camMap;
    capture >> camMap;
    detectAndDisplay(camMap);
    imshow("face detect ",camMap);
     if( waitKey( 30 ) >= 0 ){
       break;
    }
  }
}
void detectAndDisplay(Mat f){
          vector<Rect> face1,face2;
          Mat frame_gray; Mat face; Size s(300,300);
          frame_gray= f;
          cvtColor(f,frame_gray,CV_BGR2GRAY);
          face_cascade_l.detectMultiScale(frame_gray, face1, 1.1, 3, 0,Size(50,50));
          for (size_t i = 0; i< face1.size(); i++){
                    rectangle(f,face1[i],(255,255,255),2);
          }
}
```

Program 2. *OpenCV code for real time face detection*

5. CONCLUSIONS

In this thesis transform classifier based on Haar-like and LBP features are introduced and compared. The adaptive boosting classifier is a simple and efficient algorithm to train a well performance classifier. Meanwhile cascading classifier based on LBP prevailed on training processing speed and the one based on Haar won on the result of detection. The classifier trained with real face samples somehow defeated the 3D faces on the detection behavior.

However the quality of images, pose of faces and illumination are at least at present a bottleneck. Due to the huge diversity of face image in real life it is impossible to cover all condition in the training samples. These issues would also be the crucial problems needing to be solved.

REFERENCES

[1] Toan Thanh Do, Khiem Ngoc Doan, Thai Hoang Le, Bac Hoai Le, Boosted of Haar-like Features and Local Binary Pattern Based Face Detection, 2009.

Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5174627

[2] Sri-Kaushik Pavania, DavidDelgadoa, AlejandroF.Frangia, Haar-like features with optimally weighted rectangles for rapid object detection, 2010.

Available: http://www.cistib.org/gestpub/attachments/Haar-like%20features%20with%20optimally%20weighted%20rectangles%20for%20rapid%20object%20detection.pdf-7d6f1eb720e3ab5937b69a479b89ae0a.pdf

[3] Paul Viola, Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, 2001.

Available: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[4] Papageorgious C and Poggio T, A Trainable System for Object Detection, 2000.

Available: http://cbcl.mit.edu/cbcl/publications/ps/papageorgiou-poggio-IJCV-2000.pdf

[5] Rainer Lienhart and Jochen Maydt, An Extended Set of Haar-like Features for Rapid Object Detection.

Available:http://www.lienhart.de/Prof._Dr._Rainer_Lienhart/Source_Code_files/ICIP2002.pdf

[6] Yoav Freund and Robert E. Schapire, A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, 1996.

Available: http://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf

[7] Rainer Lienhart, Alexander Kuranov, Vadim Pisarevsky,Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. 2002.

Available: http://www.multimedia-computing.de/mediawiki/images/5/52/MRL-TR-May02-revised-Dec02.pdf

[8] Yoav Freund Robert E. Schapire, Experiments with a New Boosting AlgorithmMachine Learning: Proceedings of the Thirteenth International Conference, 1996.

Available: http://cseweb.ucsd.edu/~yfreund/papers/boostingexperiments.pdf

[9] Timo Ojala, Matti Pietikäinen, David Harwood, A comparative study of texture measures with classification based on featured distributions.

Available: http://www.sciencedirect.com/science/article/pii/0031320395000674

[10] Timo Ojala, Matti Pietikäinen and Topi Mäenpää, Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns.

Available: http://www.outex.oulu.fi/publications/pami_02_opm.pdf

[11] Face Detection Data Set and Benchmark Database.

Available: http://vis-www.cs.umass.edu/fddb/

[12] Personal Event Collection Database.

Availabel: https://www.vision.ee.ethz.ch/datasets_extra/pec/